NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

DISCRETE EVENT SIMULATION MODELING AND ANALYSIS OF KEY LEADER ENGAGEMENTS

by

Clifford C. Wakeman

June 2012

Thesis Co-Advisors: Arnold H. Buss
Susan M. Sanchez
Second Reader: Jason C. Caldwell

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2012	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Discrete Event Simulation Modeling and Analysis of Key Leader Engagements			**5. FUNDING NUMBERS**
6. AUTHOR(S) Clifford C. Wakeman			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ___N/A____.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

The Cultural Geography (CG) Model is a low-resolution, agent-based discrete event social simulation tailored to specific operational environments. It is based on doctrine and social theory designed to represent the behavioral response of civilian populations in conflict environments. The current version of the CG Model does not represent key leader engagements (KLE), which are activities between coalition military forces and host nation civilian personnel, as means of obtaining information, influencing behavior, and building an indigenous base of support for coalition and government objectives. These capabilities are needed for additional tactical level representation of the operational environment.

This research develops a simulation model using Simkit to explore the feasibility of modeling KLEs using discrete event simulation. A total of 32 dynamic input factors are varied using a 512-design point design. Second-order regression metamodels and partition tree models are developed for simulation model output responses that track numbers of engagements, numbers of times knowledge is provided, numbers of campaigns, and numbers of captures and kills; these analytical models are used to verify the proper execution of the simulation model. Summary statistics are analyzed to gain further insights about the simulation model's behavior.

14. SUBJECT TERMS Cultural Geography, Discrete Event Simulation, Key Leader, Key Leader Engagement, Simkit, Nearly Orthogonal and Balanced Mixed Design, Second-order Regression Metamodel, Partition Tree Model			**15. NUMBER OF PAGES** 129
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 8-98) Prescribed by ANSI Std. Z39.18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

DISCRETE EVENT SIMULATION MODELING AND ANALYSIS OF KEY LEADER ENGAGEMENTS

Clifford C. Wakeman
Captain, United States Marine Corps
B.S., University of Michigan, Ann Arbor, 2002

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
June 2012

Author: Clifford C. Wakeman

Approved by: Arnold H. Buss
Thesis Co-Advisor

Susan M. Sanchez
Thesis Co-Advisor

Jason C. Caldwell
Second Reader

Robert F. Dell
Chair, Department of Operations Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The Cultural Geography (CG) Model is a low-resolution, agent-based discrete event social simulation tailored to specific operational environments. It is based on doctrine and social theory designed to represent the behavioral response of civilian populations in conflict environments. The current version of the CG Model does not represent key leader engagements (KLE), which are activities between coalition military forces and host nation civilian personnel, as means of obtaining information, influencing behavior, and building an indigenous base of support for coalition and government objectives. These capabilities are needed for additional tactical level representation of the operational environment.

This research develops a simulation model using Simkit to explore the feasibility of modeling KLEs using discrete event simulation. A total of 32 dynamic input factors are varied using a 512-design point design. Second-order regression metamodels and partition tree models are developed for simulation model output responses that track numbers of engagements, numbers of times knowledge is provided, numbers of campaigns, and numbers of captures and kills; these analytical models are used to verify the proper execution of the simulation model. Summary statistics are analyzed to gain further insights about the simulation model's behavior.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

CG	Cultural Geography
IW	Irregular Warfare
KLE	Key Leader Engagement
NOB	Nearly Orthogonal and Balanced
NOLH	Nearly Orthogonal Latin Hypercube
OAB	Observed Attitude and Behavior
SIM	Social Impact Module
TRAC	Training and Doctrine Command Analysis Center
TWG	Tactical Wargame

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

The Cultural Geography (CG) Model, developed by TRAC-Monterey, is a low-resolution, agent-based, discrete event social simulation tailored to specific operational environments based on doctrine and social theory. It is designed to represent the behavioral responses of civilian populations in conflict environments. It focuses on the political, military, economic, social, infrastructure, and information variables in the operational environment, which affect the population's beliefs, values, interests, attitudes, and behaviors. TRAC-Monterey developed the model to support the analysis of civilian population perception based on friendly and threat actions.

The current version of the CG Model does not represent key leader engagements (KLE), which are activities between coalition military forces and host nation civilian personnel as a means of obtaining information, influencing behavior, and building an indigenous base of support for coalition and government objectives. TRAC needs this capability for additional tactical level representation of the operational environment. TRAC's Irregular Warfare (IW) Tactical Wargame (TWG) initiative utilizes Nexus, an interpretive social science simulation of IW that is separate from the CG Model, to incorporate the influence of key individuals on the population by modeling the key leader network. One of the focus areas discussed in the after-action report from the TWG that TRAC-Monterey held in October 2011 was a need to incorporate the Nexus key leader functionality into the existing CG Model. TRAC seeks to remodel the components of Nexus as discrete event simulation using Simkit, the basis for the CG Model. Currently the CG Model takes the Nexus outputs as a subset of its inputs to study a larger cultural population.

This thesis project explores three research questions. First, can we satisfactorily model KLEs using discrete event simulation and Simkit? After conducting an initial analysis of the KLE components within Nexus, we developed a discrete event simulation model that captured the critical

functionality of Nexus. This functionality includes conducting KLEs, agreeing to pass coalition force messages, honoring critical knowledge requests, campaigning by key leaders, and capturing, killing, releasing, and replacing key leaders. Additionally, we included micro-KLEs, or interactions with the general populace to extract critical knowledge. Our model involved the creation of model agents, the development of agent behaviors based primarily on an attribute called observed attitude and behavior (OAB), and the definition and development of parameters, state variables, and event graphs. We then translated the agents, behaviors, and event graphs into computer code using Java and Simkit for direct closed-loop analysis. Upon exploring the feasibility of modeling KLEs, we were able to create a simple, yet realistic, discrete event simulation model of KLEs.

Second, how can experimental design be used to assist in code verification efforts? Once complete with the discrete event simulation modeling, simulation scenarios were developed to study the KLE Model and to provide insight on what model input parameters have the greatest impact on influencing model output behaviors. Large-scale experiments were designed and employed to vary the 32 input parameters in a structured, efficient manner in order to assist with code verification efforts. Three separate scenario runtimes were used: one week to study short-term model effects, nine weeks to study the effects during a typical TWG runtime, and one year to study long-term model effects. After building regression metamodels and partition tree models for the output responses, our analysis highlighted several input factors that were important in predicting all of the output responses, such as the probability a key leader reneges from a KLE, the probability a key leader is a no-show to a KLE, and the probability a key leader honors message or knowledge requests. The identification of significant input parameters was then used to verify the proper functionality of our model by using them to explain expected behavior of the model components.

Third, are there any insights we can gain from the model using the output summary statistics coupled with histograms and boxplots, such as variability issues or outlier issues? The analysis showed that most of the output responses provide plausible ranges and variations, thus verifying the reasonableness of our model outputs. Outliers did not appear to be an issue. One output that did not behave as expected was the number of micro-KLEs response. This appeared anomalous as it exhibited exponential growth. After further investigation, we found that the results were consistent with the input parameters provided by TRAC, because a large number of potential micro-KLEs could be conducted when key leaders were unavailable.

In summary, we have built a conceptual model of the impact of key leader engagements on civilian population behavior, implemented this model using a discrete event simulation approach, and tested its performance with a large-scale experiment. This sets the stage for incorporating our KLE Model into the current CG Model, in order to improve the CG Model's suitability for use in tactical wargames and other studies.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Chapter I begins with some background about why this thesis was conducted, basically stemming from a need for key leader engagement functionality for a United States Army irregular warfare model. Next it describes what key leaders, observed attitudes and behaviors, key leader engagements, and micro-key leader engagements are. An overview of the methodology is outlined, concluding with the research questions that were posed.

A. MOTIVATION FOR THESIS

1. TRAC and the Cultural Geography Model

The United States Army Training and Doctrine Command Analysis Center, or TRAC, supports the United States Army by conducting operational analysis to inform Army decisions. TRAC-Monterey, co-located with the Naval Postgraduate School in Monterey, California, is the research and analysis arm of TRAC. It specializes in relevant, credible exploratory and applied research related to modeling, simulation, and analysis methodologies.

The Cultural Geography (CG) Model, developed by TRAC-Monterey, is a low-resolution, agent-based, discrete event social simulation tailored to specific operational environments based on doctrine and social theory. It is designed to represent the behavioral responses of civilian populations in conflict environments. It focuses on the political, military, economic, social, infrastructure, and information variables in the operational environment, which affect the population’s beliefs, values, interests, attitudes, and behaviors. TRAC-Monterey developed the model to support the analysis of civilian population perception based on friendly and threat actions.

The CG Model is built around the concept of reusable “plug-and-play” Java modules that formalize theories from behavioral and social science. It is implemented in Java and utilizes Simkit as the simulation engine. It blends a variety of carefully selected social science theories with current and emerging counterinsurgency and stability operations doctrine. It employs a social network for population entities and a bipartite network between groups and population entities to represent the evolving relationships and interactions over time. The civilian population entities and adversary entities have deep intelligence representations to allow those agents to react to events and information, and to change positions and affiliations over time with a clear understanding of motive.

2. Need for Key Leader Engagement Functionality

The current version of the CG Model does not represent key leader engagements, which are activities between coalition military forces and host nation civilian personnel as a means of obtaining information, influencing behavior, and building an indigenous base of support for coalition and government objectives. TRAC needs this capability for additional tactical level representation of the operational environment.

TRAC's Irregular Warfare (IW) Tactical Wargame (TWG) initiative utilizes Nexus, an interpretive social science simulation of IW that is separate from the CG Model, to incorporate the influence of key individuals on the population by modeling the key leader network. One of the focus areas discussed in the after-action report from the TWG that TRAC-Monterey held in October 2011 was a need to incorporate the Nexus key leader functionality into the existing CG Model. In an effort to create an integrated, simplified, and stable model that encompasses social interactions and cultural impacts, TRAC-Monterey is creating a new model, the Social Impacts Module, or SIM. The goal is to have SIM complete by the next TWG scheduled for the spring of 2013.

The Nexus Key Leader Model, a part of the Nexus suite, is a cognitive agent-based model that focuses on individual, discrete interactions among agents such as those found in key leader engagements. Nexus utilizes Repast, an agent-based modeling and simulation toolkit. Agent behaviors and symbolic interactionism are derived from interpretive social science. Agents individually adapt to civil and military intervention using Artificial Intelligence Technologies, and so they implement cultural rules using probabilistic ontologies. (Duong n.d.)

TRAC seeks to remodel the components of Nexus as discrete event simulation using Simkit, the basis for the CG Model. Currently the CG Model takes the Nexus outputs as a subset of its inputs to study a larger cultural population. This thesis project looks at the feasibility for the seamless integration of the Nexus-based code into the CG Model, thus providing improved continuity of the input parameters and the output data.

B. KEY LEADERS AND KEY LEADER ENGAGEMENTS

1. Key Leaders

Key leaders are the formal or informal leaders that are powerful in a society and can influence a target audience in a way that is beneficial for coalition operations. In the context of a TWG, key leaders are of two types. The first type is the coalition force representative, or military commander, represented by the physical player of the TWG; the human player has a simulated representation in the model. The second type is the key actor in the mission area with whom the military commander wants to engage; this is the powerbroker, stakeholder, or otherwise influential voice within the community and culture being studied, represented by a simulated entity within the model. Key leaders are one of the primary means through which players may influence the population. They can provide critical knowledge about other key leaders, threats, or resources, pass messages to the population, or inform players as to issue stances regarding community concerns.

Key leaders can be encouraged (monetarily or non-monetarily) or threatened. They can be captured or killed through player action, and if this occurs, the network of leaders within the game will reorganize through an adjudication process, and influences may change. Players begin with a unique list of known key leaders. Additional key leaders will be revealed throughout the game as the players form relationships with the population.

The motivation for key leaders to act a particular way toward coalition forces comes from an attribute called observed attitudes and behaviors (OAB). This is a key leader's general attitude toward coalition forces, either positive or negative, coupled with their propensity to act a certain way, either active or passive. The OAB types of the key leaders in this study are positive active (will go out of their way to help you), positive passive (like you but will generally stay out of the way), neutral, negative passive (do not like you but will generally stay out of the way), and negative active (will go out of their way to hurt you).

2. Key Leader Engagements

The interactions between the physical players and simulated entities are called key leader engagements (KLE). KLEs are planned to convey selected information and indicators to foreign audiences to influence their emotions, motives, objective reasoning, and ultimately the behavior of foreign governments, organizations, groups, and individuals. They are held in order to collect intelligence, develop relationships in support of commander's intent, and obtain mutually satisfying outcomes within constraints existing in a partnered nation's cultural belief system.

In general, a KLE is more than just a meeting, mini-conference, or working group between the military leaders and the local population. They are exploratory engagements in order for both sides to identify one another‘s motives. KLEs enable military leaders and decision makers to interact with key leaders and the local populace in order to begin or build relations. In addition, KLEs enable military leaders to identify the key issues and concerns of the population (McKenna and Hampsey 2010).

A subset of KLEs consists of micro-KLEs. These deal with getting information from civilians within the general population. Micro-KLEs have outcomes that are associated with the OAB of the civilian, and the civilian that is chosen to interact with is usually selected at random. Based on that person's social network, he or she might know something about a key leader, a threat, or a resource, and based on that person's motivations, he or she might tell a human player what they know. Not every micro-KLE results in useful information, and so the probability of getting actionable information is usually low.

C. RESEARCH QUESTIONS

1. Satisfactorily Modeling KLEs

Can we satisfactorily model KLEs using discrete event simulation and Simkit? Additionally, are we gaining or losing (or willing to lose) any important KLE functionality from the current method of using a third-party model? Upon exploring the feasibility of modeling KLEs, we were able to create a simple, yet realistic, discrete event simulation model of KLEs. This model also included the ability to look at micro-KLEs, a function not found within Nexus but identified by TRAC as important for SIM.

2. Significant Input Parameters and Code Verification

What input parameters are significant when predicting the model output responses? Can these significant factors assist with code verification efforts? Through the use of second-order regression metamodels and partition tree models, our analysis highlighted several input parameters that were statistically significant in predicting all of the output responses. In most cases, the metamodels and tree models backed each other up. Additionally, the factors found to be most significant helped verify the expected behavior of the model components.

3. Summary Statistic Insights

Are there any insights we can gain from the model using the output summary statistics, such as variability issues or outlier issues? The analysis showed that most of the output responses provide plausible ranges and variations, thus verifying the reasonableness of our model outputs. Outliers did not appear to be an issue. Furthermore, the summary statistics showed us that the number of micro-KLEs response appeared anomalous as it exhibited exponential growth.

D. METHODOLOGICAL APPROACH

We conducted an initial analysis of the KLE components within Nexus. The goal was to identify and understand the critical components of the network relating to KLEs. We remodeled these critical components using discrete event simulation. This involved the creation of model agents, the development of agent behaviors, and the definition and development of parameters, state variables, and event graphs. We then translated the agents, behaviors, and event graphs into computer code using Java and Simkit for direct closed-loop analysis.

Additionally, the CG Model currently uses Bayesian belief networks to model the population stance changes. Another project within TRAC-Monterey's scope is to explore the possibility of modeling the population behavior using Markov chains instead of the Bayesian belief networks. To conform to this updated population behavior methodology, Markov chains were utilized in modeling the key leader OAB changes and assignments.

Once we completed the discrete event simulation modeling, simulation scenarios were developed to study the KLE Model and to provide insight on what model input parameters have the greatest impact on influencing model output behaviors. Large-scale experiments (Kleijnen et al. 2005, Vieira et al. 2011) were designed and employed to vary the input parameters in a structured, efficient manner in order to assist with code verification efforts. Output responses similar to those in Nexus were identified, developed, and added to the KLE Model to gather information from the model for statistical analysis.

The simulation output data were collected and analyzed to identify and build any useful statistical relationships that can help predict model input outcomes. Analysis tools used included second-order regression metamodels, partition tree models, summary statistics, histograms, and boxplots.

We provide details about the KLE Model in Chapter II. In Chapter III we describe the experimental design used to investigate the KLE Model's performance. Chapter IV contains our analysis and assessment of 13 different model responses. Conclusions and suggestions for further research appear in Chapter V.

THIS PAGE INTENTIONALLY LEFT BLANK

II. KEY LEADER ENGAGEMENT MODEL

Chapter II begins with a description of the requirements needed for a closed-loop model of key leader engagements (KLE). Some of these requirements are highlighted in TRAC-Monterey supporting documentation, while others are a carryover from the Nexus KLE functionality. Next, we discuss the three types of agents used in the model, namely Blue players, Green players, and Red players, followed by behavior equations that are used to model agent behaviors. Lastly, the event graphs and components that we built are described in detail, including the component listening structure and adapters.

A. REQUIREMENTS OF KLE MODEL

Specific requirements for integrating Nexus into the Cultural Geography (CG) Model are outlined in Caldwell and Brown (2011). The model must allow agents to update their observed attitudes and behaviors (OAB), consent to pass a message, and provide critical knowledge on key leaders, threats, and/or resources. Other components are required to integrate with the CG Model, but the KLE Model in this research is run independently from the CG Model, so those functions are not explicitly implemented. The requirements document does not outline some of the KLE functionality, but it is a continuation from the legacy version of Nexus and used in comparing the KLE Model outputs to the tactical wargame (TWG) results; these functions are campaigning, capturing, killing, and replacing key leaders.

In order to model KLEs, we need to model agents, behaviors, and events. The agents represented in the model are Blue players (coalition force military commanders), Green players (key leaders), and Red players (anti-coalition force and/or anti-key leader personnel). The behaviors are represented by simple equations or probability transition matrices utilizing OABs, probability factors, probabilities, and times.

The components used in this model allow for a closed-loop execution of events that are based on discrete event simulation using Simkit. For more information on discrete event simulation modeling and discrete event programming with Simkit, see Buss (2011).

The KLE Model requirements include:

- Method to create agents;
- Method to figure out if Blue players are seeking out micro-KLEs or scheduling KLEs, to include reneges and no-shows;
- Method to handle micro-KLEs and potentially gain critical knowledge;
- Methods to handle KLEs and potentially persuade Green players to pass messages, provide critical knowledge, and/or update their OAB;
- Method to handle Green player campaigns;
- Method to handle capturing and killing of Green players;
- Method to handle releasing of Green players; and
- Method to handle Green player replacements.

B. AGENTS IN KLE MODEL

1. BluePlayer Agent

A *BluePlayer* agent in the KLE Model represents a United States military commander or coalition force commander that has the authority to conduct micro-KLEs and partake in KLEs. The agent has three attributes, summarized in Table 1. The attribute *name* is self-explanatory. The attribute *id* is a unique integer identification for the Blue player to help identify the agent in the model. The first Blue player created has an *id* of 1; each subsequent Blue player created has the next incremental integer. The attribute *incentiveToOffer* is a Boolean-type used to show if the Blue player has an incentive to offer to a Green player during a KLE.

Attribute	Type	Description
name	String	name of Blue player
id	int	integer identification of Blue player
incentiveToOffer	boolean	Blue player has (true) or does not have (false) an incentive to offer a Green player

Table 1. BluePlayer agent attributes.

2. GreenPlayer Agent

A *GreenPlayer* agent in the KLE Model represents the influential key leader. The agent has 16 attributes, summarized in Table 2. The attribute *name* is self-explanatory. The attribute *id* is a unique integer identification for the Green player to help identify the agent in the model. The first Green player created has an *id* of 1; each subsequent Green player created has the next incremental integer. The attribute *observedAttitudeBehavior* holds the current OAB for the Green player. The following are the corresponding OAB values for the representative integers: 0 is negative active, 1 is negative passive, 2 is neutral, 3 is positive passive, and 4 is positive active. The attribute *corrupt* is a Boolean-type used to show if the Green player is corrupt and will be enticed by incentives offered during KLEs. The attribute *agreedToPassMessage* is a Boolean-type used to show if the Green player has agreed to pass along a message from a Blue player during a KLE. The attribute *keyLeaderKnowledge* is a Boolean-type used to show if the Green player has critical knowledge on other key leaders to provide to a Blue player during a KLE. The attribute *threatKnowledge* is a Boolean-type used to show if the Green player has critical knowledge on threats to provide to a Blue player during a KLE. The attribute *resourceKnowledge* is a Boolean-type used to show if the Green player has critical knowledge on resources to provide to a Blue player during a KLE.

The attribute *incentivized* is a Boolean-type used to show if the Green player has been offered an incentive during a KLE. The attribute *threatened* is a Boolean-type used to show if the Green player has been presented a threat during a KLE. The attribute *killed* holds the current killed status of the Green player as an integer; 0 corresponds to alive, 1 corresponds to killed by a Blue player, and 2 corresponds to killed by a Red player. The attribute *captured* holds the current captured status of the Green player as an integer; 0 corresponds to not captured, 1 corresponds to captured by a Blue player, and 2 corresponds to captured by a Red player. The attribute *replacement* represents another *GreenPlayer* agent who is a replacement for the Green player if he is captured or killed. The attribute *kleStartTimeStamp* is used to mark the beginning of a KLE. The attribute *kleEndTimeStamp* is used to mark the end of a KLE. The attribute *campaignTimeStamp* is used to mark the start of a campaign.

Attribute	Type	Description
name	String	name of Green player
id	int	integer identification of Green player
observedAttitudeBehavior	int	Green player's observed attitude and behavior towards Blue players (0 = negative active 1 = negative passive 2 = neutral 3 = positive passive 4 = positive active)
corrupt	boolean	Green player is (true) or is not (false) corrupt
agreedToPassMessage	boolean	Green player has (true) or has not (false) agreed to pass a Blue player's message
keyLeaderKnowledge	boolean	Green player has (true) or does not have (false) critical knowledge on other key leaders
threatKnowledge	boolean	Green player has (true) or does not have (false) critical knowledge on threats
resourceKnowledge	boolean	Green player has (true) or does not have (false) critical knowledge on resources
incentivized	boolean	Green player has (true) or has not (false) been incentivized
threatened	boolean	Green player has (true) or has not (false) been threatened
killed	int	killed status of Green player (0 = alive 1 = killed by Blue player 2 = killed by Red player)
captured	int	captured status of Green player (0 = not captured 1 = captured by Blue player 2 = captured by Red player)
replacement	GreenPlayer	Green player replacement for a killed or captured Green player
kleStartTimeStamp	double	time stamp used to mark the beginning of a KLE
kleEndTimeStamp	double	time stamp used to mark the end of a KLE
campaignTimeStamp	double	time stamp used to mark the start of a campaign

Table 2. GreenPlayer agent attributes.

3. RedPlayer Agent

A *RedPlayer* agent in the KLE Model represents a person who is the enemy of the United States or coalition forces, or even of key leaders, and does not want collaboration between Blue players and Green players. A Red player could be in direct competition with a Blue player for the favor of a Green player, but this behavior is not modeled. Based on certain actions of Green players, Red players capture or kill Green players. A Red player does not have a physical representation within the model and is only referenced or implied through event names.

C. BEHAVIOR EQUATIONS IN KLE MODEL

The KLE Model uses several “behavior equations” to control certain actions by the players. These equations use simple logic to determine probabilities that players carry out a particular action. In all cases, the calculated probability or probabilities are referenced against a random uniform draw between 0 and 1 to see if the player behaves a particular way.

Behavior Equation 1 (Figure 1) is used to see if a Blue player can gain knowledge during micro-KLEs or have requests honored during KLEs. It has three variables. The first represents the OAB value, an integer between 0 and 4, of a player. The second is a random uniform draw between 0 and 1. The third is a probability factor, assumed to be between 0 and 0.2. The equation takes the OAB and adds to it the random uniform draw. The result is then multiplied by the probability factor. This gives a resulting probability that will always be between 0 and 1. The purpose of the equation is to give a range of probabilities for the player to access, and for the probabilities to be increasingly higher as the OAB value increases. For instance, if the probability factor is 0.1, a player with an OAB of 0 will have a behavior probability between 0 and 0.1. Likewise, a player with an OAB of 4 will have a behavior probability between 0.4 and 0.5. In both cases, a separate random uniform draw is compared to the probability range.

$$
\begin{aligned}
&P\{\text{Blue player gains knowledge during micro-KLE or}\\
&\quad\text{has request honored during KLE}\}\\
&= (OAB + U) * pf\\
&\\
&\quad\text{where } OAB = \text{observed attitude and behavior, between 0 and 4}\\
&\qquad\qquad U \sim Uniform(0,1)\\
&\qquad\qquad pf = \text{probability factor between 0 and 0.2}
\end{aligned}
$$

Figure 1. Behavior Equation 1

Behavior Equation 2 (Figure 2) is used to see if a Green player is going to renege from or be a no-show to a planned KLE and to see if a Blue player will offer a threat to a Green player during a KLE. It has three variables. The first represents the OAB value, an integer between 0 and 4, of a player. The second is a random uniform draw between 0 and 1. The third is a probability factor, assumed to be between 0 and 0.2. The equation subtracts four from the OAB, takes the absolute value of the result, and adds to it the random uniform draw. The result is then multiplied by the probability factor. This gives a resulting probability that will always be between 0 and 1. The purpose of the equation is to give a range of probabilities for the player to access, and for the probabilities to be decreasingly lower as the OAB value increases. For instance, if the probability factor is 0.2, a player with an OAB of 0 will have a behavior probability between 0.8 and 1. Likewise, a player with an OAB of 4 will have a behavior probability between 0 and 0.2. In both cases, a separate random uniform draw is compared to the probability range.

$$
\begin{aligned}
&P\{\text{Green player reneges from or is a no-show to a KLE, or} \\
&\quad \text{Blue player offers threat during KLE}\} \\
&= (|OAB - 4| + U) * pf \\
&\qquad \text{where } OAB = \text{observed attitude and behavior, between 0 and 4} \\
&\qquad\qquad U \sim Uniform(0,1) \\
&\qquad\qquad pf = \text{probability factor between 0 and 0.2}
\end{aligned}
$$

Figure 2. Behavior Equation 2

Behavior Equation 3 (Figure 3) is used to see if a Green player is going to be captured or killed by a Blue player following a campaign or captured or killed by a Red player following a KLE or campaign. It has two variables. The first represents a baseline probability. The second represents some amount of elapsed time between two events. The equation multiplies the baseline probability by the time. The KLE Model assumes that the resulting calculation will always be less than or equal to one to make it a valid probability, so maximum times between events need to be planned accordingly. The purpose of the equation is to give an increasing behavior probability as a player spends more time performing some action. For instance, if the baseline probability is 0.2 and a player spends 2 units of time in an activity, the player will have a behavior probability of 0.4. Then, a random uniform draw is compared to the probability.

$$
\begin{aligned}
&P\{\text{Green player is captured or killed by Blue Player after a campaign or} \\
&\quad \text{by Red player after a KLE or campaign}\} \\
&= p_{baseline} * time \\
&\qquad \text{where } p_{baseline} = \text{baseline probability between 0 and 1} \\
&\qquad\qquad time = \text{elapsed time between events}
\end{aligned}
$$

Figure 3. Behavior Equation 3

Behavior Equation 4 (Figure 4) is used to see if a Green player is going to be captured or killed by a Blue player following a KLE. It has two variables. The first represents a baseline probability. The second represents some amount of elapsed time between two events. The equation divides the baseline probability by the time. The KLE Model assumes that the resulting calculation will always be less than or equal to one to make it a valid probability, so minimum and maximum times between events need to be planned accordingly. The purpose of the equation is to give a decreasing behavior probability as a player spends more time performing some action. For instance, if the baseline probability is 0.3 and a player spends 3 units of time in an activity, the player will have a behavior probability of 0.1. Then, a random uniform draw is compared to the probability.

$$P\{\text{Green player is captured or killed by Blue Player after a KLE}\}$$
$$= \frac{p_{baseline}}{time}$$

where $p_{baseline}$ = baseline probability between 0 and 1
$time$ = elapsed time between events

Figure 4. Behavior Equation 4

Behavior Equation 5 (Figure 5), which is actually a five-by-five probability transition matrix, is used to see if a Green player updates his OAB during a KLE or after being captured, or it is used to set the OAB of a replacement after a Green player is captured or killed. The equation has three variables. The first represents the OAB value, an integer between 0 and 4, of a player. The second represents the probability of an OAB decrease. The third represents the probability of an OAB increase. The model uses the two probabilities to complete the matrix in Figure 5. For example, if the decrease probability is 0.1, the increase probability is 0.2, and the player OAB is 3, then the player will have a

0.1 probability of lowering his OAB to 2, a 0.7 probability of keeping his OAB at 3, and a 0.2 probability of raising his OAB to 4. We assume that the Green player's OAB will change by at most 1 (in either direction) after a KLE.

$$
\begin{array}{c}
\\
OAB=0\\
OAB=1\\
OAB=2\\
OAB=3\\
OAB=4
\end{array}
\begin{array}{c}
\begin{array}{ccccc}
\;\;0\;\; & \;\;1\;\; & \;\;2\;\; & \;\;3\;\; & \;\;4\;\;
\end{array}\\
\begin{bmatrix}
1-I & I & 0 & 0 & 0\\
D & 1-D-I & I & 0 & 0\\
0 & D & 1-D-I & I & 0\\
0 & 0 & D & 1-D-I & I\\
0 & 0 & 0 & D & 1-D
\end{bmatrix}
\end{array}
$$

where OAB = observed attitude and behavior

D = probability of OAB decrease

I = probability of OAB increase

Figure 5. Behavior Equation 5 probability transition matrix

D. COMPONENTS OF KLE MODEL

1. CreatePlayers

The *CreatePlayers* component creates a number of *BluePlayer* agents and *GreenPlayer* agents, each defined by the user via input parameters N_{BP} and N_{GP}, respectively, which will be used in the KLE Model. *BluePlayer* agents require a parameter p_I that gives their probability of having an incentive to offer. *GreenPlayer* agents require four parameters, p_C, p_{KLK}, p_{TK}, and p_{RK}, which give probabilities for being corrupt, having key leader critical knowledge, having threat critical knowledge, and having resource critical knowledge, respectively.

Parameters for the *CreatePlayers* component are summarized in Table 3.

Parameter	Description
N_{BP}	total number of Blue players
p_I	probability of Blue player having incentive to offer
N_{GP}	total number of Green players
p_C	probability of Green player being corrupt
p_{KLK}	probability of Green player having key leader knowledge
p_{TK}	probability of Green player having threat knowledge
p_{RK}	probability of Green player having resource knowledge

Table 3. CreatePlayers parameters

The event graph for the *CreatePlayers* component is shown in Figure 6.

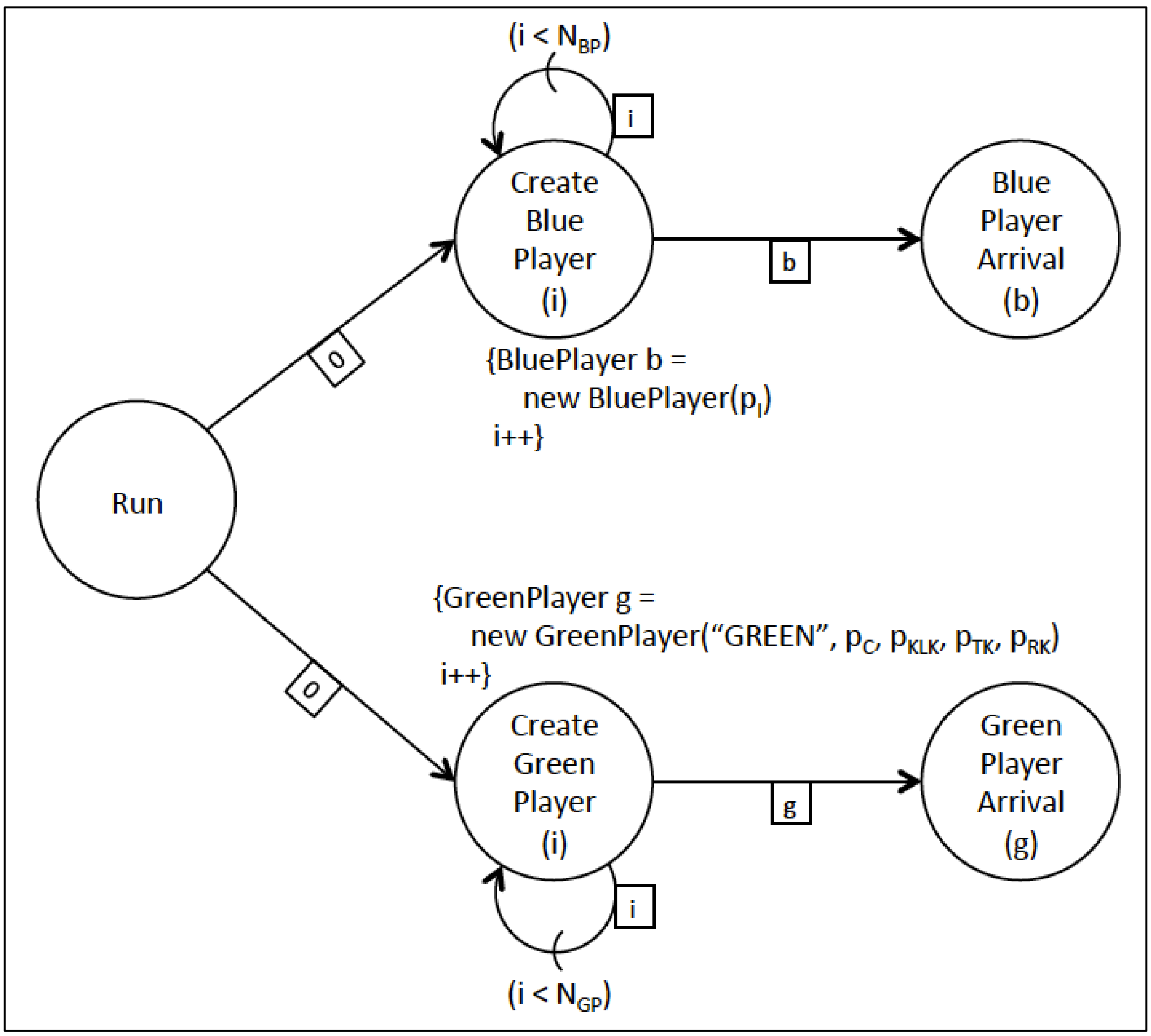


Figure 6. *CreatePlayers* event graph

The *Run* event schedules the *CreateBluePlayer* and *CreateGreenPlayer* events, passing the local parameter zero to both.

The *CreateBluePlayer* and *CreateGreenPlayer* events simulate adding a Blue player or Green player, respectively, to the model. They each take in a local integer parameter to keep track of how many players have been created. Each event creates a *BluePlayer* or *GreenPlayer* agent, respectively, increments the local integer parameter by one, and schedules a *BluePlayerArrival* or *GreenPlayerArrival* event, respectively, passing along the created agent. The self-scheduling loops schedule another agent creation if the local integer variable is less than the parameters N_{BP} or N_{GP}, respectively.

The *BluePlayerArrival* and *GreenPlayerArrival* events each simulate a Blue player or Green player, respectively, looking to schedule their first KLE. They take in a local parameter represented by a *BluePlayer* or *GreenPlayer* agent, respectively.

2. HandleEngagementType

The *HandleEngagementType* component handles the scheduling of micro-KLEs and KLEs. It has six input parameters. It requires four random distributions representing the stream of times that Blue players schedule their next micro-KLE ($\{t_{NM}\}$), the stream of times that Blue Players schedule their next KLE ($\{t_{NK}\}$), the stream of times that Green players renege from a KLE ($\{t_{RG}\}$), and the stream of times that Blue players schedule their next arrival for another micro-KLE or KLE ($\{t_{BM}\}$). The parameter pf_{RG}, which is a number between 0 and 0.2, is used as a probability factor to calculate whether a Green player is going to renege from a KLE. The parameter pf_{NS}, which is a number between 0 and 0.2, is used as a probability factor to calculate whether a Green player is a no-show to a KLE.

The *HandleEngagementType* component has two state variables that represent lists; q is a queue to hold the arriving Green players to the component, and x is a list of any Green players that have been canceled and no longer needed in the model.

Parameters, parameter constraints, and state variables for the *HandleEngagementType* component are summarized in Table 4.

Parameter	Description
$\{t_{NM}\}$	stream of Blue player next micro-KLE times
$\{t_{NK}\}$	stream of Blue player next KLE times
$\{t_{RG}\}$	stream of Green player renege times
$\{t_{BM}\}$	stream of Blue player next meeting times
pf_{RG}	renege probability factor
pf_{NS}	no-show probability factor
Parameter Constraint	
$0 \leq pf_{RG} \leq 0.2$	
$0 \leq pf_{NS} \leq 0.2$	
State Variable	**Description**
q	queue to hold arriving Green players
x	list of canceled Green players

Table 4. *HandleEngagementType* parameters, parameter constraints, and state variables

The event graph for the *HandleEngagementType* component is shown in Figure 7.

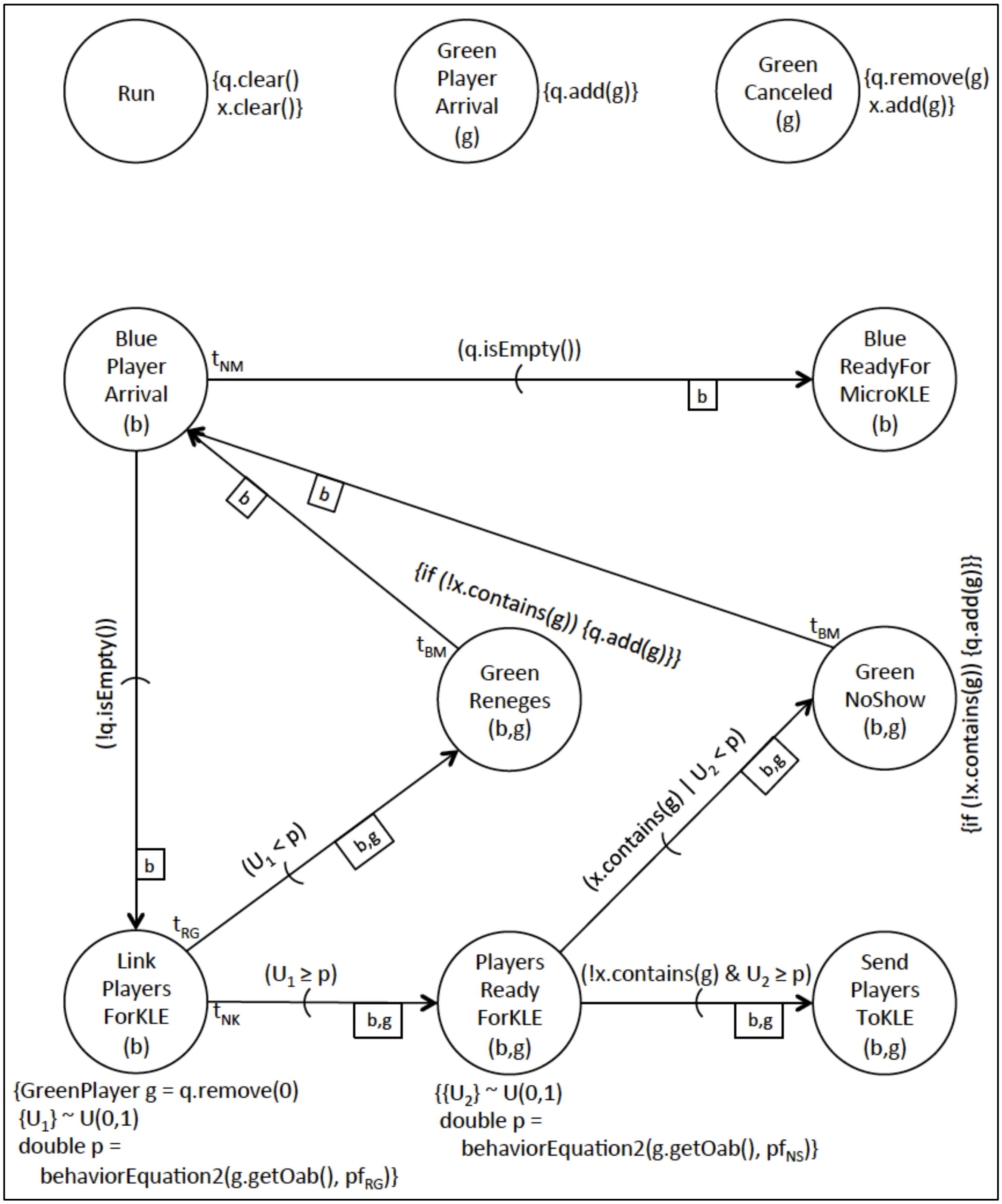


Figure 7. *HandleEngagementType* event graph

The *Run* event clears *q* and *x*.

The *BluePlayerArrival* event simulates a Blue player looking for a micro-KLE or looking to set up a KLE. It takes a *BluePlayer* agent as its local parameter. If there are no Green players to meet, a *BlueReadyForMicroKLE* event is scheduled with a time delay pulled from $\{t_{NM}\}$, passing along the local Blue player. If there is a Green player available, a *LinkPlayersForKLE* event is scheduled, passing along the local Blue player.

The *GreenPlayerArrival* event simulates a Green player looking to set up a KLE. It takes a *GreenPlayer* agent as its local parameter. It adds the local Green player to *q*.

The *BlueReadyForMicroKLE* event simulates a Blue player being ready to start a micro-KLE. It takes a *BluePlayer* agent as its local parameter.

The *LinkPlayersForKLE* event simulates the initial agreement by a Blue player and Green player to set up a KLE. It takes a *BluePlayer* agent as its local parameter. Since the model assumes Blue players have no preference for which Green player they engage, it removes the first Green player from *q* and assigns it to a local *GreenPlayer* agent variable. It draws a random uniform number between 0 and 1. It then calculates the probability that the local Green player reneges by using the Green player's OAB value and the parameter pf_{RG} in behavior Equation 2 (Figure 2). If the random uniform draw is less than the calculated renege probability, it schedules a *GreenReneges* event with a time delay pulled from $\{t_{RG}\}$, passing along the local Blue player and local Green player. If the random uniform draw is greater than or equal to the calculated renege probability, it schedules a *PlayersReadyForKLE* event with a time delay pulled from $\{t_{NK}\}$, passing along the local Blue player and local Green player.

The *GreenReneges* event simulates a Green player calling off a planned KLE. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. If the local Green player is not in *x*, it adds the Green player back to *q*. If the Green player is not canceled, the assumption is that the Green player is still alive and has reneged. If the Green player is canceled, the assumption is that the Green player is not alive or no longer available, and the Blue player is made aware of this fact before showing up for the KLE. This event schedules a *BluePlayerArrival* event with a time delay pulled from $\{t_{BM}\}$, passing along the local Blue player.

The *PlayersReadyForKLE* event checks if a Blue player and Green player are ready to start a KLE. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It draws a random uniform number between 0 and 1. It then calculates the probability that the local Green player is a no-show by using the Green player's OAB value and the parameter pf_{NS} in behavior Equation 2 (Figure 2). If the local Green player is in x, or if the random uniform draw is less than the calculated no-show probability, it schedules a *GreenNoShow* event, passing along the local Blue player and local Green player. If the local Green player is not in x and the random uniform draw is greater than or equal to the calculated no-show probability, it schedules a *SendPlayersToKLE* event, passing along the local Blue player and local Green player.

The *GreenNoShow* event simulates a Green player not showing up for a KLE. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. If the local Green player is not in x, it adds the Green player back to q. If the Green player is not canceled, the assumption is that the Green player is still alive and is a no-show. If the Green player is canceled, the assumption is that the Green player is not alive or no longer available, and the Blue player is made aware of this fact upon showing up for the KLE. This event schedules a *BluePlayerArrival* event with a time delay pulled from $\{t_{BM}\}$, passing along the local Blue player.

The *SendPlayersToKLE* event simulates a Blue player and Green player being ready to start a KLE. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters.

The *GreenCanceled* event simulates a Green player who is a replacement being no longer needed in the model. It takes a *GreenPlayer* agent as its local parameter. It removes the local Green player from q, and it adds the player to x.

3. MicroKeyLeaderEngagement

The *MicroKeyLeaderEngagement* component represents a micro-KLE with an entity that is not a key leader. It has two input parameters. It requires a random distribution representing the stream of times that Blue players will spend in a micro-KLE ($\{t_M\}$). The parameter pf_{CK}, which is a number between 0 and 0.2, is used as a probability factor to calculate whether a Blue player is going to gain critical knowledge during a micro-KLE.

The *MicroKeyLeaderEngagement* component has two state variables. The variable N_{MKLE} tracks the number of micro-KLEs held. The variable N_{TKG} tracks the number of times critical knowledge is gained from a micro-KLE.

Parameters, parameter constraints, and state variables for the *MicroKeyLeaderEngagement* component are summarized in Table 5.

Parameter	**Description**
$\{t_M\}$	stream of micro-KLE times
pf_{CK}	chance of knowledge probability factor
Parameter Constraint	
$0 \le pf_{CK} \le 0.2$	
State Variable	**Description**
N_{MKLE}	number of micro-KLEs (initialized to 0)
N_{TKG}	number of times critical knowledge gained (initialized to 0)

Table 5. *MicroKeyLeaderEngagement* parameters, parameter constraints, and state variables

The event graph for the *MicroKeyLeaderEngagement* component is shown in Figure 8.

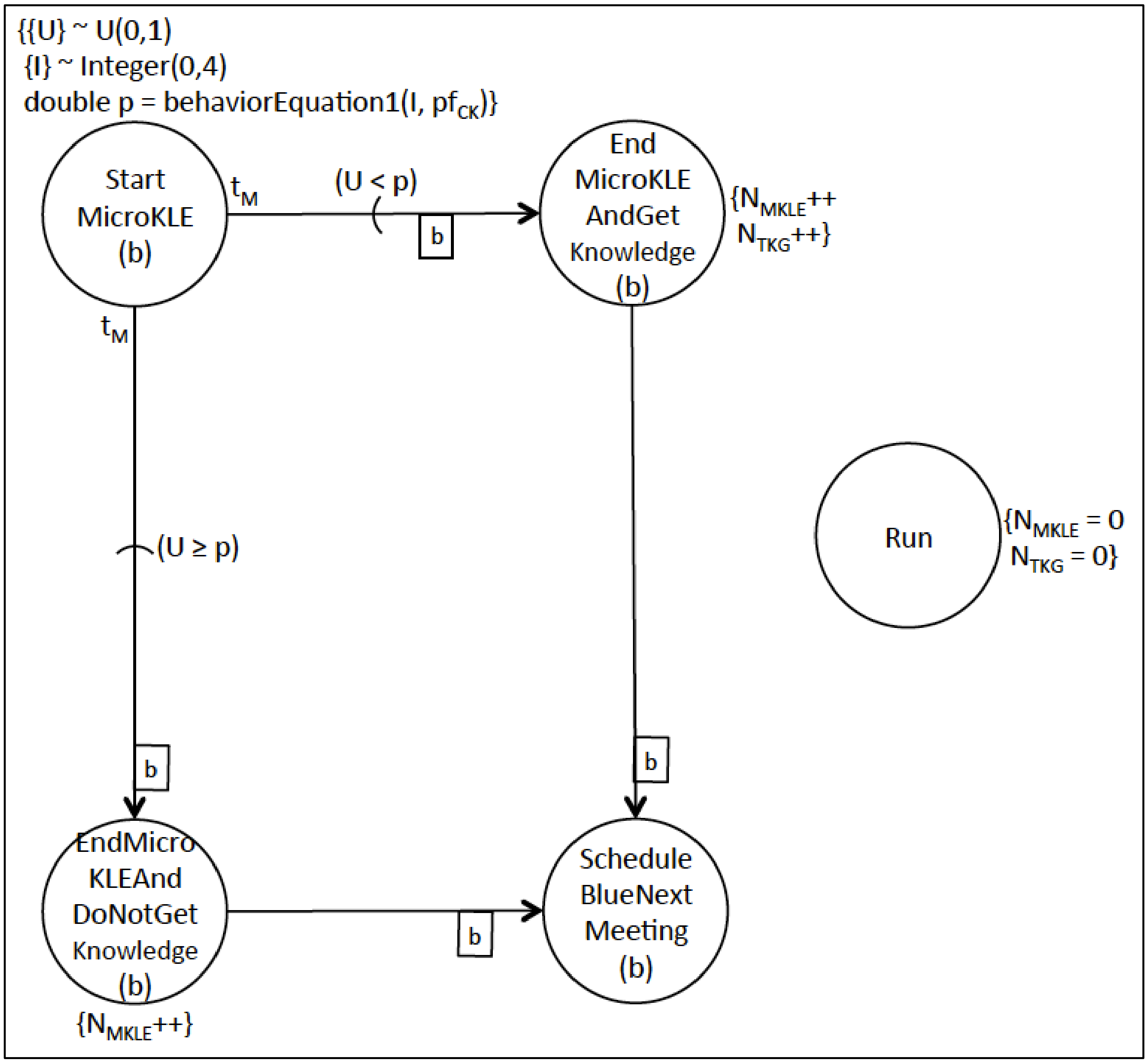


Figure 8. *MicroKeyLeaderEngagement* event graph

The *Run* event initializes the two state variables to zero.

The *StartMicroKLE* event simulates the beginning of a micro-KLE. It takes a *BluePlayer* agent as its local parameter. It draws a random uniform number between 0 and 1. It also draws a random integer between 0 and 4 that represents the OAB of the non-key leader. It then calculates the probability that the non-key leader honors the local Blue player's critical knowledge request by using the random integer draw and the parameter pf_{CK} in behavior Equation 1 (Figure 1). If the random uniform draw is less than the calculated knowledge probability, it schedules an *EndMicroKLEAndGetKnowledge* event with a time delay pulled from {t_M}, passing along the local Blue player. If the random uniform draw is greater than or equal to the calculated knowledge probability, it schedules an *EndMicroKLEAndDoNotGetKnowledge* event with the same time delay and by passing the Blue player.

The *EndMicroKLEAndGetKnowledge* event simulates the end of a micro-KLE and a Blue player getting critical knowledge. It takes a *BluePlayer* agent as its local parameter. It increments N_{MKLE} and N_{TKG} both by one. It then schedules a *ScheduleBlueNextMeeting* event, passing along the local Blue player.

The *EndMicroKLEAndDoNotGetKnowledge* event simulates the end of a micro-KLE and a Blue player not getting any critical knowledge. It takes a *BluePlayer* agent as its local parameter. It increments N_{MKLE} by one. It then schedules a *ScheduleBlueNextMeeting*, passing along the local Blue player.

The S*cheduleBlueNextMeeting* event simulates a Blue player scheduling his next arrival for a micro-KLE or KLE. It takes a *BluePlayer* agent as its local parameter.

4. KeyLeaderEngagement

The *KeyLeaderEngagement* component represents a KLE occurrence. It has three input parameters. It requires two random distributions representing the stream of times that Blue players and Green players will spend in a KLE ({t_K}) and the stream of times that Blue players schedule their next arrival for another micro-KLE or KLE ({t_{BM}}). The parameter p_I is the same parameter from the *CreatePlayers* component representing the probability that a Blue player has an incentive to offer.

The *KeyLeaderEngagement* component has two state variables. The variable N_{KLE} tracks the number of KLEs held. The variable x is a list of any Green players that have been canceled and no longer needed in the model.

Parameters and state variables for the *KeyLeaderEngagement* component are summarized in Table 6.

Parameter	Description
$\{t_K\}$	stream of KLE times
$\{t_{BM}\}$	stream of Blue player next meeting times
p_I	probability of Blue player having incentive to offer
State Variable	**Description**
N_{KLE}	number of KLEs (initialized to 0)
x	list of canceled Green players

Table 6. *KeyLeaderEngagement* parameters and state variables

The event graph for the *KeyLeaderEngagement* component is shown in Figure 9.

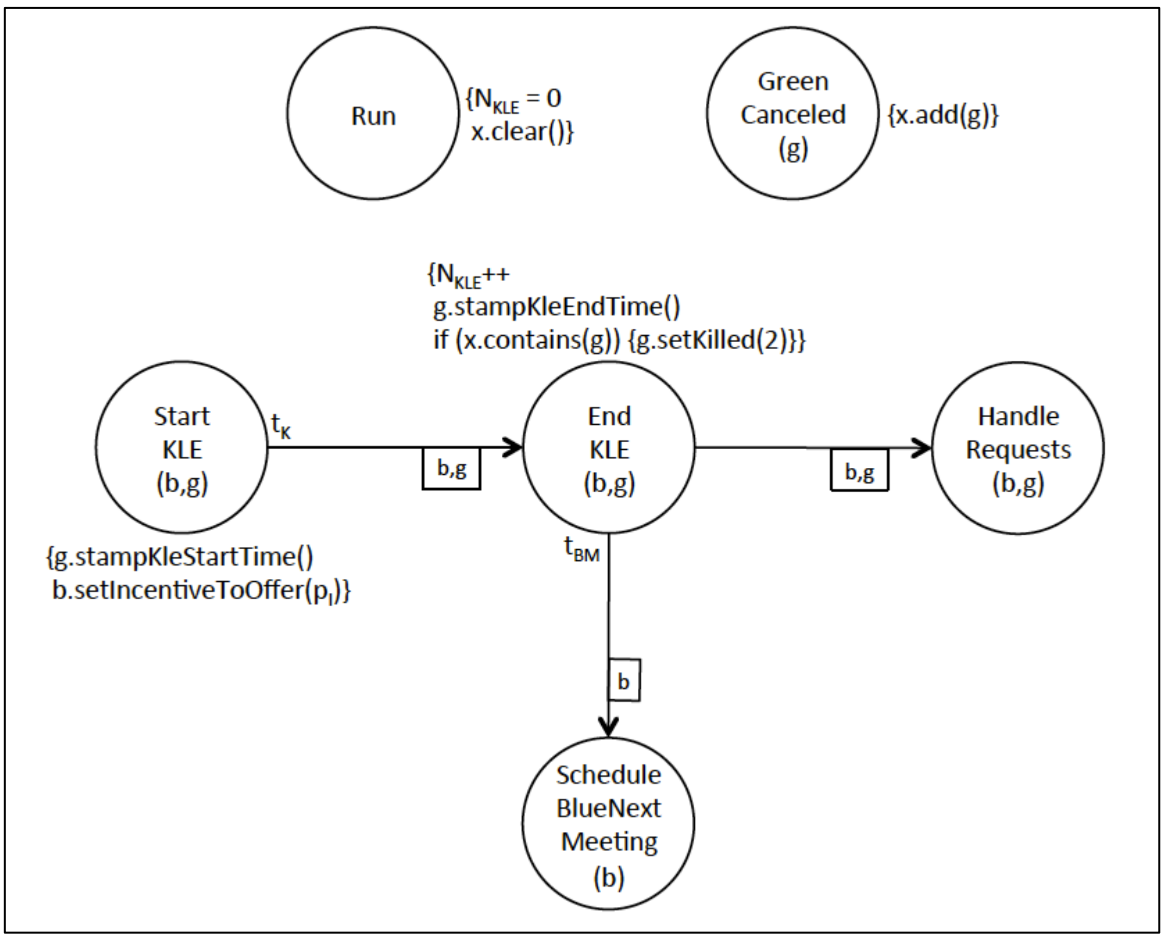


Figure 9. *KeyLeaderEngagement* event graph

The *Run* event initializes N_{KLE} to zero. It also clears x.

The *StartKLE* event simulates the beginning of a KLE. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It stamps the KLE start time for the local Green player. It resets whether the local Blue player has an incentive to offer using p_I. Lastly, it schedules an *EndKLE* event with a time delay pulled from $\{t_K\}$, passing along the local Blue player and local Green player.

The *EndKLE* event simulates the end of a KLE. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It increments N_{KLE} by one, stamps the KLE end time for the local Green player, and, if the local Green player is in x, sets the killed status of the local Green player as if he was killed by a Red player. This event schedules a *ScheduleBlueNextMeeting* event with a time delay pulled from $\{t_{BM}\}$, passing along the local Blue player. It also schedules a *HandleRequests* event, passing along the local Blue player and local Green player.

The S*cheduleBlueNextMeeting* event simulates a Blue player scheduling his next arrival for a micro-KLE or KLE. It takes a *BluePlayer* agent as its local parameter.

The *HandleRequests* event simulates a Blue player and Green player going over the KLE requests. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters.

The *GreenCanceled* event simulates a Green player replacement who is no longer needed in the model. It takes a *GreenPlayer* agent as its local parameter. It adds the local Green player to x.

5. HandleMessageRequest

The *HandleMessageRequest* component represents a *GreenPlayer* agent deciding if he will pass a message from a *BluePlayer* agent to those under his influence. It has four input parameters. The parameter pf_{BT}, which is a number between 0 and 0.2, is used as a probability factor to calculate whether a Blue player will offer a threat during a KLE to get the Green player to pass a message. The final three parameters, pf_{HR}, pf_{HRI}, and pf_{HRT}, all between 0 and 0.2, are used as probability factors to calculate whether a Green player will honor the Blue player's request outright, honor with an incentive, or honor with a threat, respectively.

The *HandleMessageRequest* component has one state variable. The variable N_{HRM} tracks the number of honored message requests.

Parameters, parameter constraints, and state variables for the *HandleMessageRequest* component are summarized in Table 7.

Parameter	**Description**
pf_{BT}	Blue player offering threat probability factor
pf_{HR}	honor request probability factor
pf_{HRI}	honor request with incentive probability factor
pf_{HRT}	honor request with threat probability factor
Parameter Constraint	
$0 \le pf_{BT} \le 0.2$	
$0 \le pf_{HR} \le 0.2$	
$0 \le pf_{HRI} \le 0.2$	
$0 \le pf_{HRT} \le 0.2$	
State Variable	**Description**
N_{HRM}	number of honored message requests (initialized to 0)

Table 7. *HandleMessageRequest* parameters, parameter constraints, and state variables

The event graph for the *HandleMessageRequest* component is shown in Figures 10 and 11.

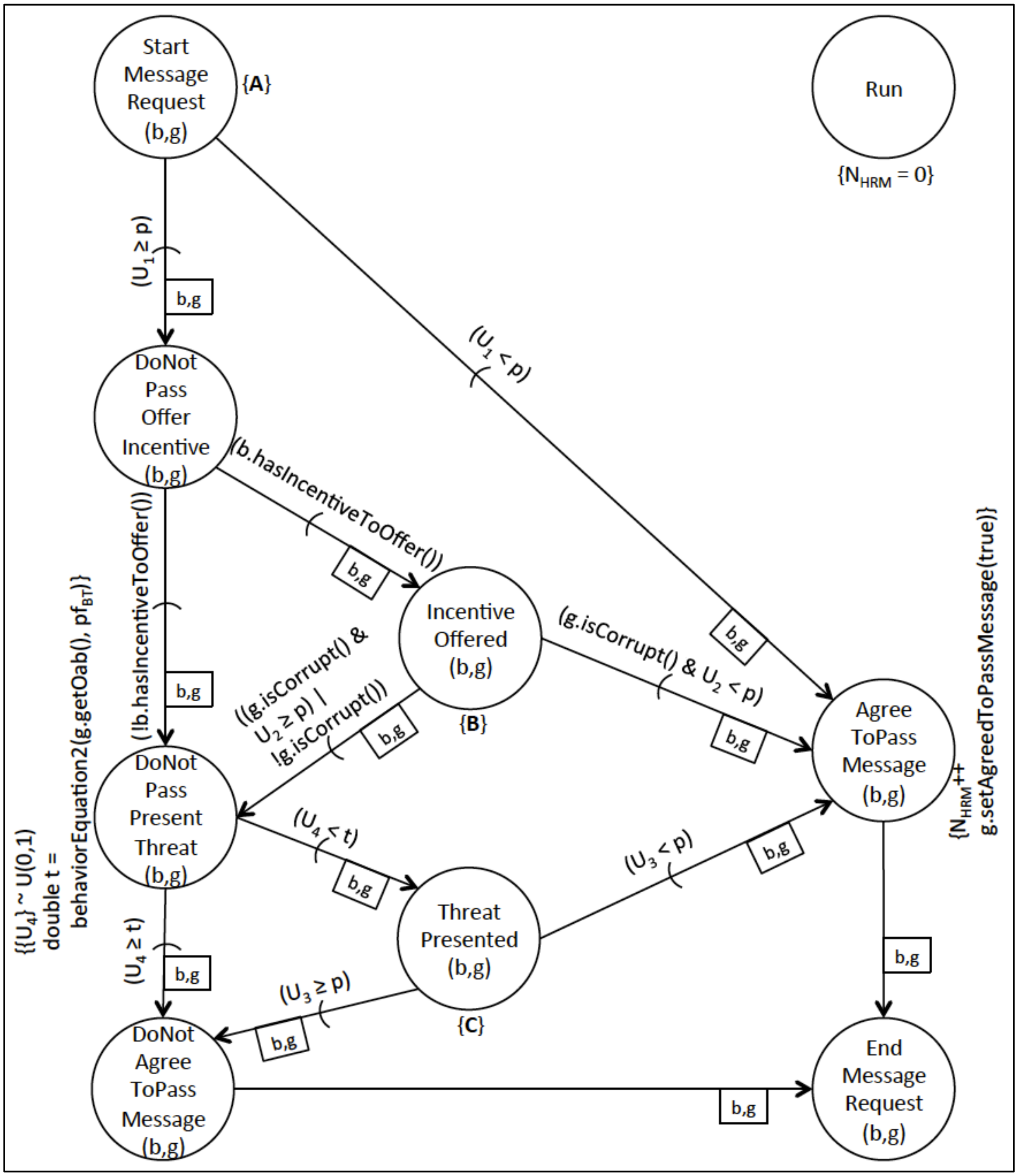


Figure 10. *HandleMessageRequest* event graph (part 1)

A

{g.setAgreedToPassMessage(false)
{U_1} ~ U(0,1)
double p = behaviorEquation1(g.getOab(), pf_{HR})}

B

{g.setIncentivized(true)
{U_2} ~ U(0,1)
double p = behaviorEquation1(g.getOab(), pf_{HRI})}

C

{g.setThreatened(true)
{U_3} ~ U(0,1)
double p = behaviorEquation1(g.getOab(), pf_{HRT})}}

Figure 11. *HandleMessageRequest* event graph (part 2)

The *Run* event initializes N_{HRM} to zero.

The *StartMessageRequest* event simulates the beginning of the message request. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It resets the fact that the local Green player has agreed to pass a message to false. Next, it draws a random uniform number between 0 and 1. It then calculates the probability that the local Green player will honor the request outright to pass a message by using the Green player's OAB value and the parameter pf_{HR} in behavior Equation 1 (Figure 1). If the random uniform draw is less than the calculated honoring request probability, it schedules an *AgreeToPassMessage* event, passing along the local Blue player and local Green player. If the random uniform draw is greater than or equal to the calculated honoring request probability, it schedules a *DoNotPassOfferIncentive* event, passing along the local Blue player and local Green player.

The *AgreeToPassMessage* event simulates a Green player agreeing to pass a message. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It increments N_{HRM} by one, and it sets the fact that the local Green player has agreed to pass a message to true. It schedules an *EndMessageRequest* event, passing along the local Blue player and local Green player.

The *DoNotPassOfferIncentive* event simulates a Green player deciding not to pass a message and a Blue player potentially offering an incentive to persuade the Green player to change his mind and pass a message. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. If the local Blue player has an incentive to offer to the local Green player, it schedules an *IncentiveOffered* event, passing along the local Blue player and local Green player. If the local Blue player does not have an incentive to offer, it schedules a *DoNotPassPresentThreat* event, passing along the local Blue player and local Green player.

The *IncentiveOffered* event simulates a Blue player offering an incentive to a Green player to persuade him to pass a message. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It sets the fact that the local Green player has been incentivized to true. Next, it draws a random uniform number between 0 and 1. It then calculates the probability that the local Green player will honor the request to pass a message given an incentive by using the Green player's OAB value and the parameter pf_{HRI} in behavior Equation 1 (Figure 1). If the Green player is corrupt and the random uniform draw is less than the calculated honoring request probability, it schedules an *AgreeToPassMessage* event, passing along the local Blue player and local Green player. If the Green player is corrupt and the random uniform draw is greater than or equal to the calculated honoring request probability, or if the Green player is not corrupt, it schedules a *DoNotPassPresentThreat* event, passing along the local Blue player and local Green player.

The *DoNotPassPresentThreat* event simulates a Green player deciding not to pass a message and a Blue player potentially presenting a threat to persuade the Green player to change his mind and pass a message. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It draws a random uniform number between 0 and 1. It then calculates the probability that the local Blue player will threaten the local Green player by using the Green player's OAB value and the parameter pf_{BT} in behavior Equation 2 (Figure 2). If the random uniform draw is less than the calculated threat probability, it schedules a *ThreatPresented* event, passing along the local Blue player and local Green player. If the random uniform draw is greater than or equal to the calculated threat probability, it schedules a *DoNotAgreeToPassMessage* event, passing along the local Blue player and local Green player.

The *ThreatPresented* event simulates a Blue player threatening a Green player to persuade him to pass a message. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It sets the fact that the local Green player has been threatened to true. Next, it draws a random uniform number between 0 and 1. It then calculates the probability that the local Green player will honor the request to pass a message given a threat by using the Green player's OAB value and the parameter pf_{HRT} in behavior Equation 1 (Figure 1). If the random uniform draw is less than the calculated honoring request probability, it schedules an *AgreeToPassMessage* event, passing along the local Blue player and local Green player. If the random uniform draw is greater than or equal to the calculated honoring request probability, it schedules a *DoNotAgreeToPassMessage* event, passing along the local Blue player and local Green player.

The *DoNotAgreeToPassMessage* event simulates a Green player ultimately not agreeing to pass a message. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It schedules an *EndMessageRequest* event, passing along the local Blue player and local Green player.

The *EndMessageRequest* event simulates the end of the message request. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters.

6. HandleKeyLeaderKnowledgeRequest

The *HandleKeyLeaderKnowledgeRequest* component represents a *GreenPlayer* agent deciding on whether to provide key leader critical knowledge to a *BluePlayer*. It has five input parameters. The parameters pf_{BT}, pf_{HR}, pf_{HRI}, and pf_{HRT} are the same as those used in the *HandleMessageRequest* component. The same parameter constraints apply to these four parameters. The parameter p_{KLK} is the same parameter from the *CreatePlayers* component representing the probability that a Green player has key leader critical knowledge.

The *HandleKeyLeaderKnowledgeRequest* component has one state variable. The variable N_{HRK} tracks the number of honored key leader knowledge requests.

Parameters, parameter constraints, and state variables for the *HandleKeyLeaderKnowledgeRequest* component are summarized in Table 8.

Parameter	Description
pf_{BT}	Blue player offering threat probability factor
pf_{HR}	honor request probability factor
pf_{HRI}	honor request with incentive probability factor
pf_{HRT}	honor request with threat probability factor
p_{KLK}	probability of Green player having key leader knowledge
Parameter Constraint	
$0 \le pf_{BT} \le 0.2$	
$0 \le pf_{HR} \le 0.2$	
$0 \le pf_{HRI} \le 0.2$	
$0 \le pf_{HRT} \le 0.2$	
State Variable	**Description**
N_{HRK}	number of honored key leader knowledge requests (initialized to 0)

Table 8. *HandleKeyLeaderKnowledgeRequest* parameters, parameter constraints, and state variables

The event graph for the *HandleKeyLeaderKnowledgeRequest* component is shown in Figures 12 and 13.

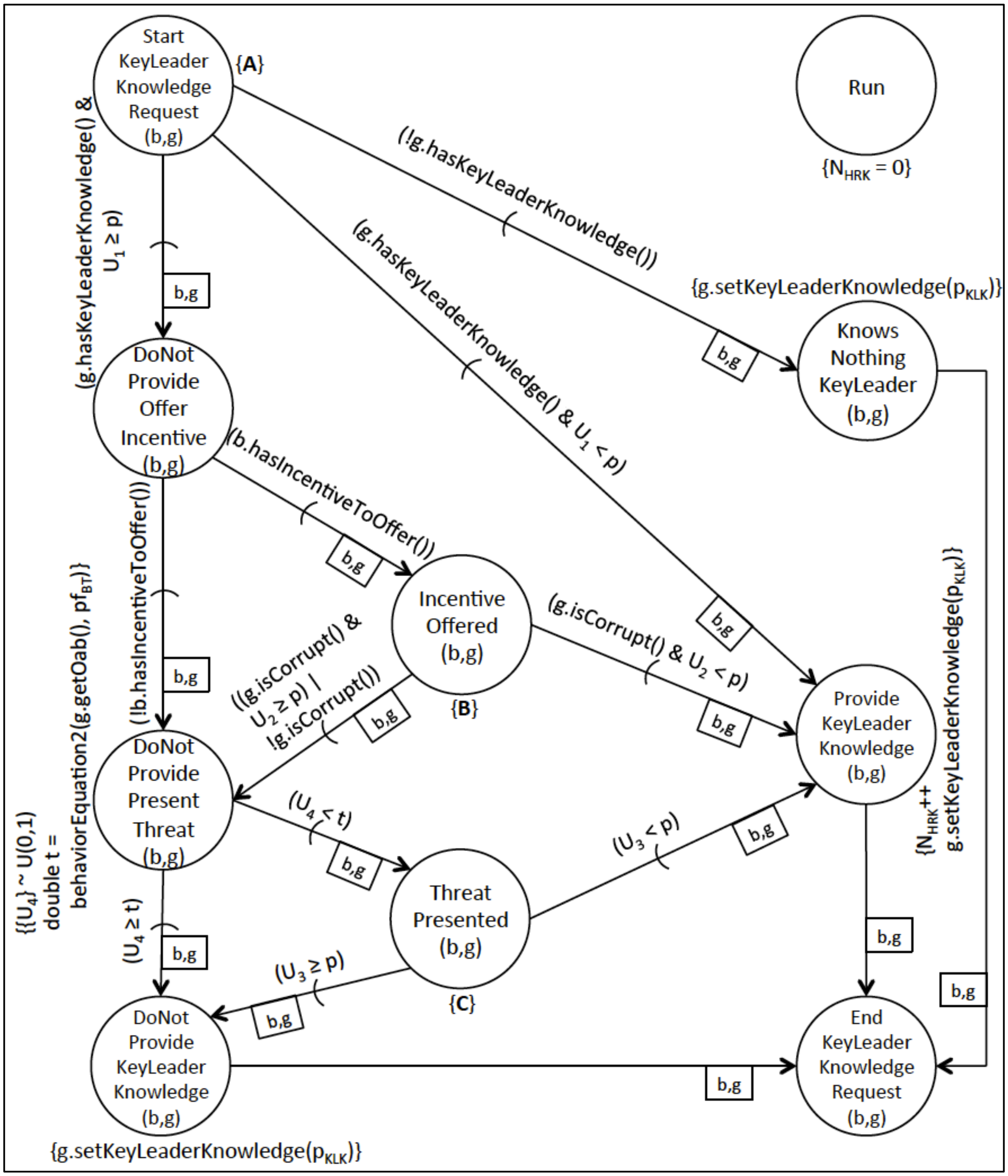


Figure 12. *HandleKeyLeaderKnowledgeRequest* event graph (part 1)

A

$\{\{U_1\} \sim U(0,1)$

double p = behaviorEquation1(g.getOab(), $pf_{HR})\}$

B

{g.setIncentivized(true)

$\{U_2\} \sim U(0,1)$

double p = behaviorEquation1(g.getOab(), $pf_{HRI})\}\}$

C

{g.setThreatened(true)

$\{U_3\} \sim U(0,1)$

double p = behaviorEquation1(g.getOab(), $pf_{HRT})\}\}$

Figure 13. *HandleKeyLeaderKnowledgeRequest* event graph (part 2)

The *Run* event initializes N_{HRK} to zero.

The *StartKeyLeaderKnowledgeRequest* event simulates the beginning of the key leader critical knowledge request. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It draws a random uniform number between 0 and 1. It then calculates the probability that the local Green player will honor the request outright to provide key leader critical knowledge by using the Green player's OAB value and the parameter pf_{HR} in behavior Equation 1 (Figure 1). If the local Green player does not have any key leader critical knowledge, it schedules a *KnowsNothingKeyLeader* event, passing along the local Blue player and local Green player. If the Green player has key leader knowledge and the random uniform draw is less than the calculated honoring request probability, it schedules a *ProvideKeyLeaderKnowledge* event, passing along the local Blue player and local Green player. If the Green player has key leader knowledge and the random uniform draw is greater than or equal to the calculated honoring request probability, it schedules a *DoNotProvideOfferIncentive* event, passing along the local Blue player and local Green player.

The *KnowsNothingKeyLeader* event simulates a Green player not having any knowledge on other key leaders. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It reinitializes whether the local Green player has key leader critical knowledge by using the parameter p_{KLK}. It schedules an *EndKeyLeaderKnowledgeReques*t event, passing along the local Blue player and local Green player.

The *ProvideKeyLeaderKnowledge* event simulates a Green player providing key leader critical knowledge. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It increments N_{HRK} by one, and it resets whether the local Green player has key leader critical knowledge by using the parameter p_{KLK}. It schedules an *EndKeyLeaderKnowledgeReques*t event, passing along the local Blue player and local Green player.

The *DoNotProvideOfferIncentive* event simulates a Green player deciding not to provide key leader critical knowledge and a Blue player potentially offering an incentive to get such knowledge. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. If the local Blue player has an incentive to offer to the local Green player, it schedules an *IncentiveOffered* event, passing along the local Blue player and local Green player. If the local Blue player does not have an incentive to offer, it schedules a *DoNotProvidePresentThreat* event, passing along the local Blue player and local Green player.

The *IncentiveOffered* event simulates a Blue player offering an incentive to a Green player in an attempt to extract key leader critical knowledge. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It sets the fact that the local Green player has been incentivized to true. Next, it draws a random uniform number between 0 and 1. It then calculates the probability that the local Green player will honor the request to provide key leader critical knowledge given an incentive by using the Green player's OAB value and the parameter pf_{HRI} in behavior Equation 1 (Figure 1). If the Green player is corrupt and the random uniform draw is less than the calculated honoring request probability, it schedules a *ProvideKeyLeaderKnowledge* event, passing along the local Blue player and local Green player. If the Green player is corrupt and the random uniform draw is greater than or equal to the calculated honoring request probability, or if the Green player is not corrupt, it schedules a *DoNotProvidePresentThreat* event, passing along the local Blue player and local Green player.

The *DoNotProvidePresentThreat* event simulates a Green player deciding not to provide key leader critical knowledge and a Blue player potentially presenting a threat to get such knowledge. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It draws a random uniform number between 0 and 1. It then calculates the probability that the local Blue player will threaten the local Green player by using the Green player's OAB value and the parameter pf_{BT} in behavior Equation 2 (Figure 2). If the random uniform draw is less than the calculated threat probability, it schedules a *ThreatPresented* event, passing along the local Blue player and local Green player. If the random uniform draw is greater than or equal to the calculated threat probability, it schedules a *DoNotProvideKeyLeaderKnowledge* event, passing along the local Blue player and local Green player.

The *ThreatPresented* event simulates a Blue player threatening a Green player for key leader critical knowledge. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It sets the fact that the local Green player has been threatened to true. Next, it draws a random uniform number between 0 and 1. It then calculates the probability that the local Green player will honor the request to provide key leader critical knowledge given a threat by using the Green player's OAB value and the parameter pf_{HRT} in behavior Equation 1 (Figure 1). If the random uniform draw is less than the calculated honoring request probability, it schedules a *ProvideKeyLeaderKnowledge* event, passing along the local Blue player and local Green player. If the random uniform draw is greater than or equal to the calculated honoring request probability, it schedules a *DoNotProvideKeyLeaderKnowledge* event, passing along the local Blue player and local Green player.

The *DoNotProvideKeyLeaderKnowledge* event simulates a Green player ultimately not providing key leader critical knowledge. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It resets whether the local Green player has key leader critical knowledge by using the parameter p_{KLK}. It then schedules an *EndKeyLeaderKnowledgeRequest* event, passing along the local Blue player and local Green player.

The *EndKeyLeaderKnowledgeRequest* event simulates the end of the key leader critical knowledge request. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters.

7. HandleThreatKnowledgeRequest

The *HandleThreatKnowledgeRequest* component represents a *GreenPlayer* agent deciding on whether to provide threat critical knowledge to a *BluePlayer*. It has five input parameters. The parameters pf_{BT}, pf_{HR}, pf_{HRI}, and pf_{HRT} are the same as those used in the *HandleMessageRequest* component. The same parameter constraints apply to these four parameters. The parameter p_{TK} is the same parameter from the *CreatePlayers* component representing the probability that a Green player has threat critical knowledge.

The *HandleThreatKnowledgeRequest* component has one state variable. The variable N_{HRT} tracks the number of honored threat knowledge requests.

Parameters, parameter constraints, and state variables for the *HandleThreatKnowledgeRequest* component are summarized in Table 9.

Parameter	Description
pf_{BT}	Blue player offering threat probability factor
pf_{HR}	honor request probability factor
pf_{HRI}	honor request with incentive probability factor
pf_{HRT}	honor request with threat probability factor
p_{TK}	probability of Green player having threat knowledge
Parameter Constraint	
$0 \le pf_{BT} \le 0.2$	
$0 \le pf_{HR} \le 0.2$	
$0 \le pf_{HRI} \le 0.2$	
$0 \le pf_{HRT} \le 0.2$	
State Variable	**Description**
N_{HRT}	number of honored threat knowledge requests (initialized to 0)

Table 9. *HandleThreatKnowledgeRequest* parameters, parameter constraints, and state variables

The event graph for the *HandleThreatKnowledgeRequest* component is shown in Figures 14 and 15.

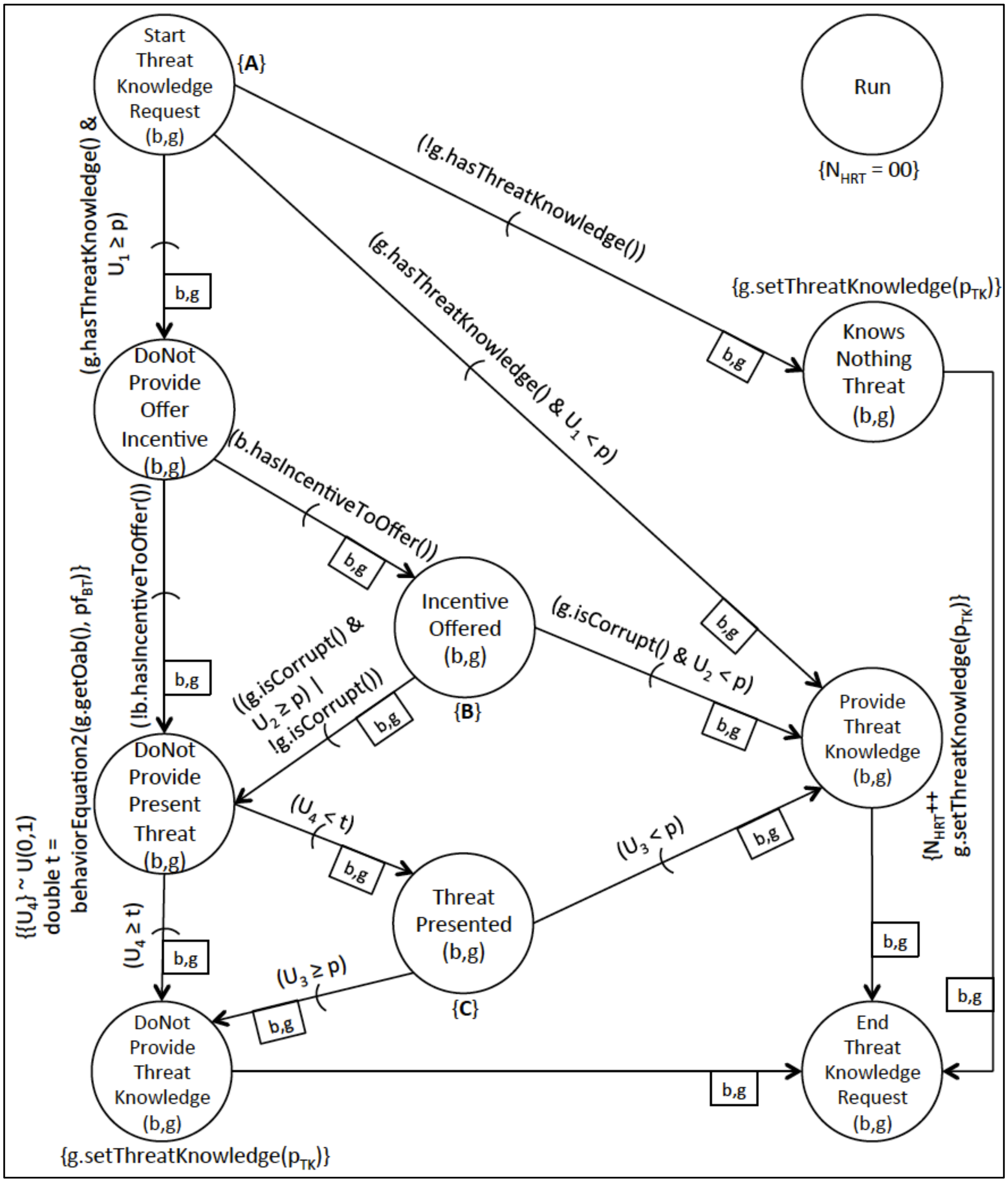


Figure 14. *HandleThreatKnowledgeRequest* event graph (part 1)

A

{{U_1} ~ U(0,1)

double p = behaviorEquation1(g.getOab(), pf_{HR})}

B

{g.setIncentivized(true)

{U_2} ~ U(0,1)

double p = behaviorEquation1(g.getOab(), pf_{HRI})}}

C

{g.setThreatened(true)

{U_3} ~ U(0,1)

double p = behaviorEquation1(g.getOab(), pf_{HRT})}}

Figure 15. *HandleThreatKnowledgeRequest* event graph (part 2)

The *Run* event initializes N_{HRT} to zero.

The *StartThreatKnowledgeRequest* event simulates the beginning of the threat critical knowledge request. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It draws a random uniform number between 0 and 1. It then calculates the probability that the local Green player will honor the request outright to provide threat critical knowledge by using the Green player's OAB value and the parameter pf_{HR} in behavior Equation 1 (Figure 1). If the local Green player does not have any threat critical knowledge, it schedules a *KnowsNothingThreat* event, passing along the local Blue player and local Green player. If the Green player has threat knowledge and the random uniform draw is less than the calculated honoring request probability, it schedules a *ProvideThreatKnowledge* event, passing along the local Blue player and local Green player. If the Green player has threat knowledge and the random uniform draw is greater than or equal to the calculated honoring request probability, it schedules a *DoNotProvideOfferIncentive* event, passing along the local Blue player and local Green player.

The *KnowsNothingThreat* event simulates a Green player not having any knowledge on threats. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It reinitializes whether the local Green player has threat critical knowledge by using the parameter p_{TK}. It schedules an *EndThreatKnowledgeRequest* event, passing along the local Blue player and local Green player.

The *ProvideThreatKnowledge* event simulates a Green player providing threat critical knowledge. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It increments N_{HRT} by one, and it resets whether the local Green player has threat critical knowledge by using the parameter p_{TK}. It schedules an *EndThreatKnowledgeRequest* event, passing along the local Blue player and local Green player.

The *DoNotProvideOfferIncentive* event simulates a Green player deciding not to provide threat critical knowledge and a Blue player potentially offering an incentive to get such knowledge. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. If the local Blue player has an incentive to offer to the local Green player, it schedules an *IncentiveOffered* event, passing along the local Blue player and local Green player. If the local Blue player does not have an incentive to offer, it schedules a *DoNotProvidePresentThreat* event, passing along the local Blue player and local Green player.

The *IncentiveOffered* event simulates a Blue player offering an incentive to a Green player in an attempt to extract threat critical knowledge. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It sets the fact that the local Green player has been incentivized to true. Next, it draws a random uniform number between 0 and 1. It then calculates the probability that the local Green player will honor the request to provide threat critical knowledge given an incentive by using the Green player's OAB value and the parameter pf_{HRI} in behavior Equation 1 (Figure 1). If the Green player is corrupt and the random uniform draw is less than the calculated honoring request probability, it schedules a *ProvideThreatKnowledge* event, passing along the local Blue player and local

Green player. If the Green player is corrupt and the random uniform draw is greater than or equal to the calculated honoring request probability, or if the Green player is not corrupt, it schedules a *DoNotProvidePresentThreat* event, passing along the local Blue player and local Green player.

The *DoNotProvidePresentThreat* event simulates a Green player deciding not to provide threat critical knowledge and a Blue player potentially presenting a threat to get such knowledge. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It draws a random uniform number between 0 and 1. It then calculates the probability that the local Blue player will threaten the local Green player by using the Green player's OAB value and the parameter pf_{BT} in behavior Equation 2 (Figure 2). If the random uniform draw is less than the calculated threat probability, it schedules a *ThreatPresented* event, passing along the local Blue player and local Green player. If the random uniform draw is greater than or equal to the calculated threat probability, it schedules a *DoNotProvideThreatKnowledge* event, passing along the local Blue player and local Green player.

The *ThreatPresented* event simulates a Blue player threatening a Green player for threat critical knowledge. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It sets the fact that the local Green player has been threatened to true. Next, it draws a random uniform number between 0 and 1. It then calculates the probability that the local Green player will honor the request to provide threat critical knowledge given a threat by using the Green player's OAB and the parameter pf_{HRT} in behavior Equation 1 (Figure 1). If the random uniform draw is less than the calculated honoring request probability, it schedules a *ProvideThreatKnowledge* event, passing along the local Blue player and local Green player. If the random uniform draw is greater than or equal to the calculated honoring request probability, it schedules a *DoNotProvideThreatKnowledge* event, passing along the local Blue player and local Green player.

The *DoNotProvideThreatKnowledge* event simulates a Green player ultimately not providing threat critical knowledge. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It resets whether the local Green player has threat critical knowledge by using the parameter p_{TK}. It then schedules an *EndThreatKnowledgeRequest* event, passing along the local Blue player and local Green player.

The *EndThreatKnowledgeRequest* event simulates the end of the threat critical knowledge request. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters.

8. HandleResourceKnowledgeRequest

The *HandleResourceKnowledgeRequest* component represents a *GreenPlayer* agent deciding on whether to provide resource critical knowledge to a *BluePlayer*. It has five input parameters. The parameters pf_{BT}, pf_{HR}, pf_{HRI}, and pf_{HRT} are the same as those used in the *HandleMessageRequest* component. The same parameter constraints apply to these four parameters. The parameter p_{RK} is the same parameter from the *CreatePlayers* component representing the probability that a Green player has resource critical knowledge.

The *HandleResourceKnowledgeRequest* component has one state variable. The variable N_{HRR} tracks the number of honored resource knowledge requests.

Parameters, parameter constraints, and state variables for the *HandleResourceKnowledgeRequest* component are summarized in Table 10.

Parameter	Description
pf_{BT}	Blue player offering threat probability factor
pf_{HR}	honor request probability factor
pf_{HRI}	honor request with incentive probability factor
pf_{HRT}	honor request with threat probability factor
p_{RK}	probability of Green player having resource knowledge
Parameter Constraint	
$0 \le pf_{BT} \le 0.2$	
$0 \le pf_{HR} \le 0.2$	
$0 \le pf_{HRI} \le 0.2$	
$0 \le pf_{HRT} \le 0.2$	
State Variable	**Description**
N_{HRR}	number of honored resource knowledge requests (initialized to 0)

Table 10. *HandleResourceKnowledgeRequest* parameters, parameter constraints, and state variables

The event graph for the *HandleResourceKnowledgeRequest* component is shown in Figures 16 and 17.

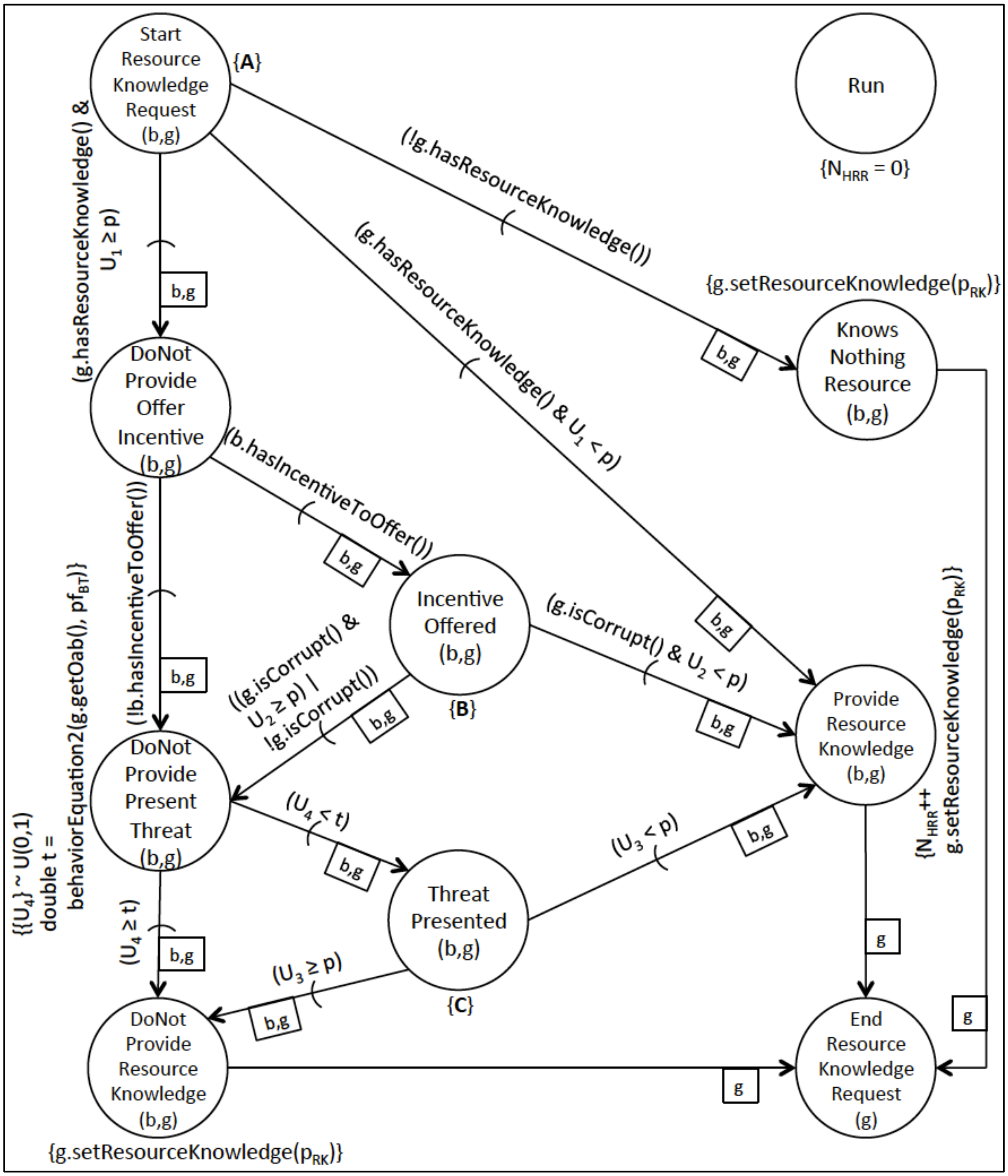


Figure 16. *HandleResourceKnowledgeRequest* event graph (part 1)

A
{{U_1} ~ U(0,1)
double p = behaviorEquation1(g.getOab(), pf_{HR})}

B
{g.setIncentivized(true)
{U_2} ~ U(0,1)
double p = behaviorEquation1(g.getOab(), pf_{HRI})}}

C
{g.setThreatened(true)
{U_3} ~ U(0,1)
double p = behaviorEquation1(g.getOab(), pf_{HRT})}}

Figure 17. *HandleResourceKnowledgeRequest* event graph (part 2)

The *Run* event initializes N_{HRR} to zero.

The *StartResourceKnowledgeRequest* event simulates the beginning of the resource critical knowledge request. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It draws a random uniform number between 0 and 1. It then calculates the probability that the local Green player will honor the request outright to provide resource critical knowledge by using the Green player's OAB value and the parameter pf_{HR} in behavior Equation 1 (Figure 1). If the local Green player does not have any resource critical knowledge, it schedules a *KnowsNothingResource* event, passing along the local Blue player and local Green player. If the Green player has resource knowledge and the random uniform draw is less than the calculated honoring request probability, it schedules a *ProvideResourceKnowledge* event, passing along the local Blue player and local Green player. If the Green player has resource knowledge and the random uniform draw is greater than or equal to the calculated honoring request probability, it schedules a *DoNotProvideOfferIncentive* event, passing along the local Blue player and local Green player.

The *KnowsNothingResource* event simulates a Green player not having any knowledge on resources. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It reinitializes whether the local Green player has resource critical knowledge by using the parameter p_{RK}. It schedules an *EndResourceKnowledgeRequest* event, passing along the local Green player.

The *ProvideResourceKnowledge* event simulates a Green player providing resource critical knowledge. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It increments N_{HRR} by one, and it resets whether the local Green player has resource critical knowledge by using the parameter p_{RK}. It schedules an *EndResourceKnowledgeRequest* event, passing along the local Green player.

The *DoNotProvideOfferIncentive* event simulates a Green player deciding not to provide resource critical knowledge and a Blue player potentially offering an incentive to get such knowledge. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. If the local Blue player has an incentive to offer to the local Green player, it schedules an *IncentiveOffered* event, passing along the local Blue player and local Green player. If the local Blue player does not have an incentive to offer, it schedules a *DoNotProvidePresentThreat* event, passing along the local Blue player and local Green player.

The *IncentiveOffered* event simulates a Blue player offering an incentive to a Green player in an attempt to extract resource critical knowledge. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It sets the fact that the local Green player has been incentivized to true. Next, it draws a random uniform number between 0 and 1. It then calculates the probability that the local Green player will honor the request to provide resource critical knowledge given an incentive by using the Green player's OAB value and the parameter pf_{HRI} in behavior Equation 1 (Figure 1). If the Green player is corrupt and the random uniform draw is less than the calculated honoring request probability, it schedules a *ProvideResourceKnowledge* event, passing along the local Blue player and local Green player. If the Green player is corrupt and the random uniform draw is greater than or equal to the calculated honoring request probability, or if the Green player is not corrupt, it schedules a *DoNotProvidePresentThreat* event, passing along the local Blue player and local Green player.

The *DoNotProvidePresentThreat* event simulates a Green player deciding not to provide resource critical knowledge and a Blue player potentially presenting a threat to get such knowledge. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It draws a random uniform number between 0 and 1. It then calculates the probability that the local Blue player will threaten the local Green player by using the Green player's OAB value and the parameter pf_{BT} in behavior Equation 2 (Figure 2). If the random uniform draw is less than the calculated threat probability, it schedules a *ThreatPresented* event, passing along the local Blue player and local Green player. If the random uniform draw is greater than or equal to the calculated threat probability, it schedules a *DoNotProvideResourceKnowledge* event, passing along the local Blue player and local Green player.

The *ThreatPresented* event simulates a Blue player threatening a Green player for resource critical knowledge. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It sets the fact that the local Green player has been threatened to true. Next, it draws a random uniform number between 0 and 1. It then calculates the probability that the local Green player will honor the request to provide resource critical knowledge given a threat by using the Green player's OAB value and the parameter pf_{HRT} in behavior Equation 1 (Figure 1). If the random uniform draw is less than the calculated honoring request probability, it schedules a *ProvideResourceKnowledge* event, passing along the local Blue player and local Green player. If the random uniform draw is greater than or equal to the calculated honoring request probability, it schedules a *DoNotProvideResourceKnowledge* event, passing along the local Blue player and local Green player.

The *DoNotProvideResourceKnowledge* event simulates a Green player ultimately not providing resource critical knowledge. It takes both a *BluePlayer* and *GreenPlayer* agent as its local parameters. It resets whether the local Green player has resource critical knowledge by using the parameter p_{RK}. It then schedules an *EndResourceKnowledgeRequest* event, passing along the local Green player.

The *EndResourceKnowledgeRequest* event simulates the end of the resource critical knowledge request. It takes a *GreenPlayer* agent as its local parameter.

9. UpdateOAB

The *UpdateOAB* component handles the updating of a Green player's OAB depending on what happens during a KLE. It has eight input parameters. The parameters p_{D0} and p_{I0} represent the probabilities of an OAB decrease or increase, respectively, given the Green player not being incentivized and not being threatened; the sum of these two must be less than or equal to 1. The parameters p_{DI} and p_{II} represent the probabilities of an OAB decrease or increase, respectively, given the Green player being incentivized and not being threatened; the sum of these two must be less than or equal to 1. The parameters p_{DT} and p_{IT} represent the probabilities of an OAB decrease or increase, respectively, given the Green player being threatened; the sum of these two must be less than or equal to 1. The parameters p_{BCKB} and p_{BCKR} are baseline probabilities of a Green player being captured or killed by either a Blue player or Red player, respectively. In order to avoid overlapping probability ranges, one minus p_{BCKR} times the maximum possible KLE time must be greater than or equal to p_{BCKB}.

Parameters and parameter constraints for the *UpdateOAB* component are summarized in Table 11.

Parameter	Description
p_{D0}	probability of OAB decrease when Green player not incentivized and not threatened
p_{I0}	probability of OAB increase when Green player not incentivized and not threatened
p_{DI}	probability of OAB decrease when Green player incentivized and not threatened
p_{II}	probability of OAB increase when Green player incentivized and not threatened
p_{DT}	probability of OAB decrease when Green player threatened
p_{IT}	probability of OAB increase when Green player threatened
p_{BCKB}	baseline probability of Green player captured or killed by Blue player
p_{BCKR}	baseline probability of Green player captured or killed by Red player
Parameter Constraint	
$p_{D0} + p_{I0} \leq 1$	
$p_{DI} + p_{II} \leq 1$	
$p_{DT} + p_{IT} \leq 1$	
$p_{BCKB} \leq 1 - p_{BCKR}$ * (max KLE time)	

Table 11. *UpdateOAB* parameters and parameter constraints

The event graph for the *UpdateOAB* component is shown in Figure 18.

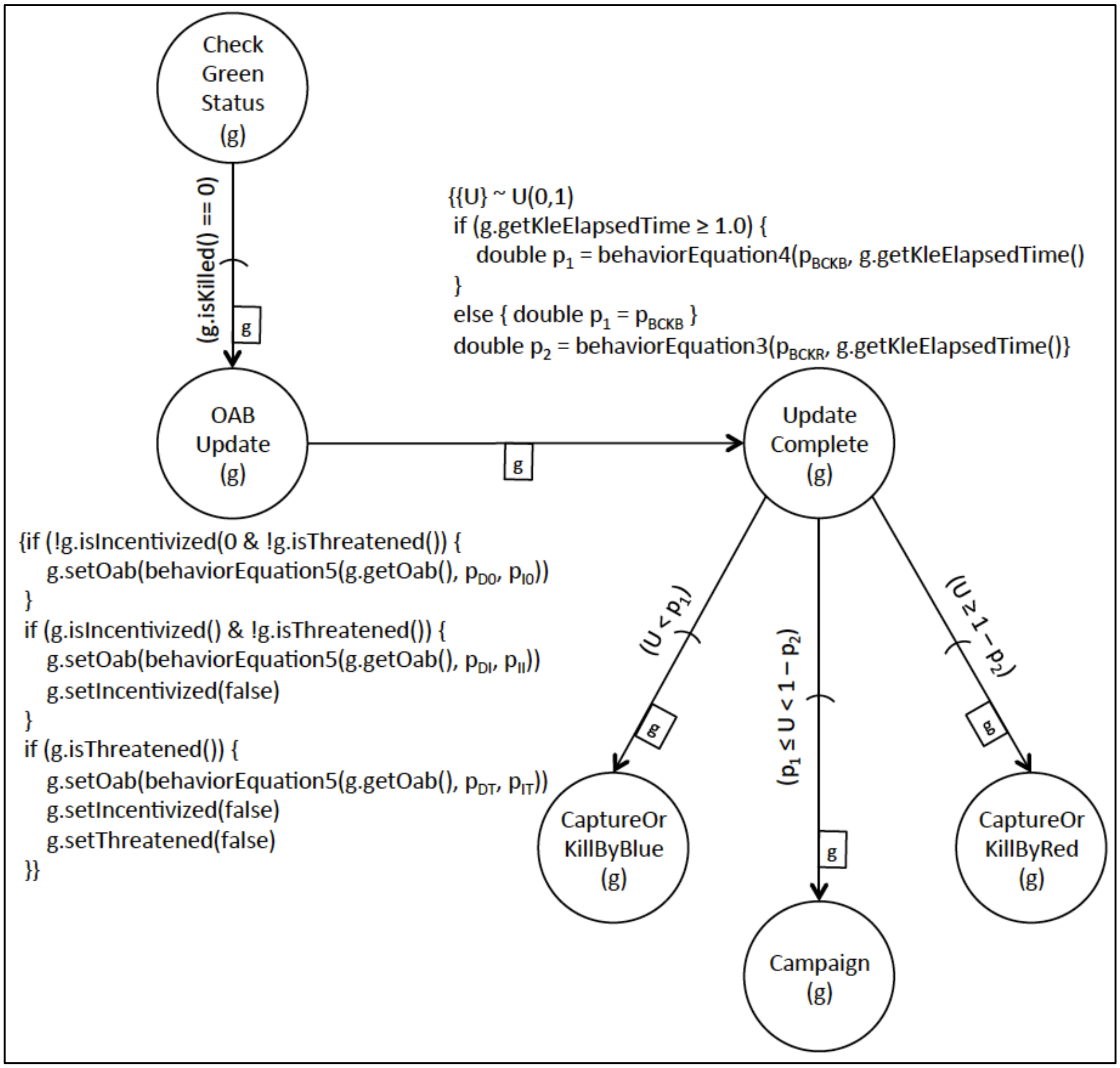


Figure 18. *UpdateOAB* event graph

The *CheckGreenStatus* event checks if a Green player is still in the model following a KLE. It takes a *GreenPlayer* agent as its local parameter. If the local Green player has not been canceled during a KLE (not killed in this case), it schedules an *OABUpdate* event, passing along the local Green player.

The *OABUpdate* event simulates a Green player changing his OAB after a KLE. It takes a *GreenPlayer* agent as its local parameter. If the Green player has not been incentivized or threatened during the KLE, it calculates the Green player's new OAB by using his current OAB, D = p_{D0}, and I = p_{I0} in behavior

Equation 5 (Figure 5). If the Green player has been incentivized but not threatened during the KLE, it calculates the Green player's new OAB by using his current OAB, D = p_{DI}, and I = p_{II} in behavior Equation 5 (Figure 5). It then resets the fact that the Green player is incentivized to false. If the Green player has been threatened during the KLE, it calculates the Green player's new OAB by using his current OAB, D = p_{DT}, and I = p_{IT} in behavior Equation 5 (Figure 5). It then resets the facts that the Green player is incentivized and threatened to false. Lastly, it schedules an *UpdateComplete* event, passing along the local Green player.

The *UpdateComplete* event simulates a Green player completing his OAB update. It takes a *GreenPlayer* agent as its local parameter. It draws a random uniform number between 0 and 1. If the time that the Green player spent in the KLE is greater than or equal to one, it calculates the probability that he is captured or killed by a Blue player by using p_{BCKB} and the time spent in the KLE in behavior Equation 4 (Figure 4). If the time that the Green player spent in the KLE is less than one, the probability of being captured or killed by a Blue player equals p_{BCKB}. It also calculates the probability that the local Green player is captured or killed by a Red player by using p_{BCKR} and the time spent in the KLE in behavior Equation 3 (Figure 3). If the random uniform draw is less than the calculated capture or kill by Blue probability, it schedules a *CaptureOrKillByBlue* event, passing along the local Green player. If the random uniform draw is greater than or equal to one minus the calculated capture or kill by Red probability, it schedules a *CaptureOrKillByRed* event, passing along the local Green player. If the random uniform draw is greater than or equal to the calculated capture or kill by Blue probability and less than one minus the calculated capture or kill by Red probability, it schedules a *Campaign* event, passing along the local Green player.

The *Campaign* event simulates a Green player looking to campaign. It takes a *GreenPlayer* agent as its local parameter.

The *CaptureOrKillByBlue* event simulates a Green player being captured or killed by a Blue player. It takes a *GreenPlayer* agent as its local parameter.

The *CaptureOrKillByRed* event simulates a Green player being captured or killed by a Red player. It takes a *GreenPlayer* agent as its local parameter.

10. Campaign

The *Campaign* component handles whether a Green player will campaign following a KLE. It has five input parameters. It requires three random distributions representing the stream of times that Green players schedule their next campaign ($\{t_{NC}\}$), the stream of times that Green players spend campaigning ($\{t_C\}$), and the stream of times that Green players schedule their next arrival for another KLE ($\{t_{GM}\}$). The parameters p_{BCKB} and p_{BCKR} are the same as defined in the *UpdateOAB* component. Additional constraints on these two parameters are that p_{BCKB} times the maximum campaign time and p_{BCKR} times the maximum campaign time both must be less than or equal to one; this ensures that probabilities greater than one are not encountered.

The *Campaign* component has two state variables. The variable N_{PC} tracks the number of pro-coalition force campaigns. The variable N_{AC} tracks the number of anti-coalition force campaigns.

Parameters, parameter constraints, and state variables for the *Campaign* component are summarized in Table 12.

Parameter	Description
$\{t_{NC}\}$	stream of Green player next campaign times
$\{t_C\}$	stream of Green player campaign times
$\{t_{GM}\}$	stream of Green player next meeting times
p_{BCKB}	baseline probability of Green player captured or killed by Blue player
p_{BCKR}	baseline probability of Green player captured or killed by Red player
Parameter Constraint	
p_{BCKB} * (max campaign time) ≤ 1	
p_{BCKR} * (max campaign time) ≤ 1	
State Variable	**Description**
N_{PC}	number of pro-CF campaigns (initialized to 0)
N_{AC}	number of anti-CF campaigns (initialized to 0)

Table 12. *Campaign* parameters, parameter constraints, and state variables

The event graph for the *Campaign* component is shown in Figures 19 and 20.

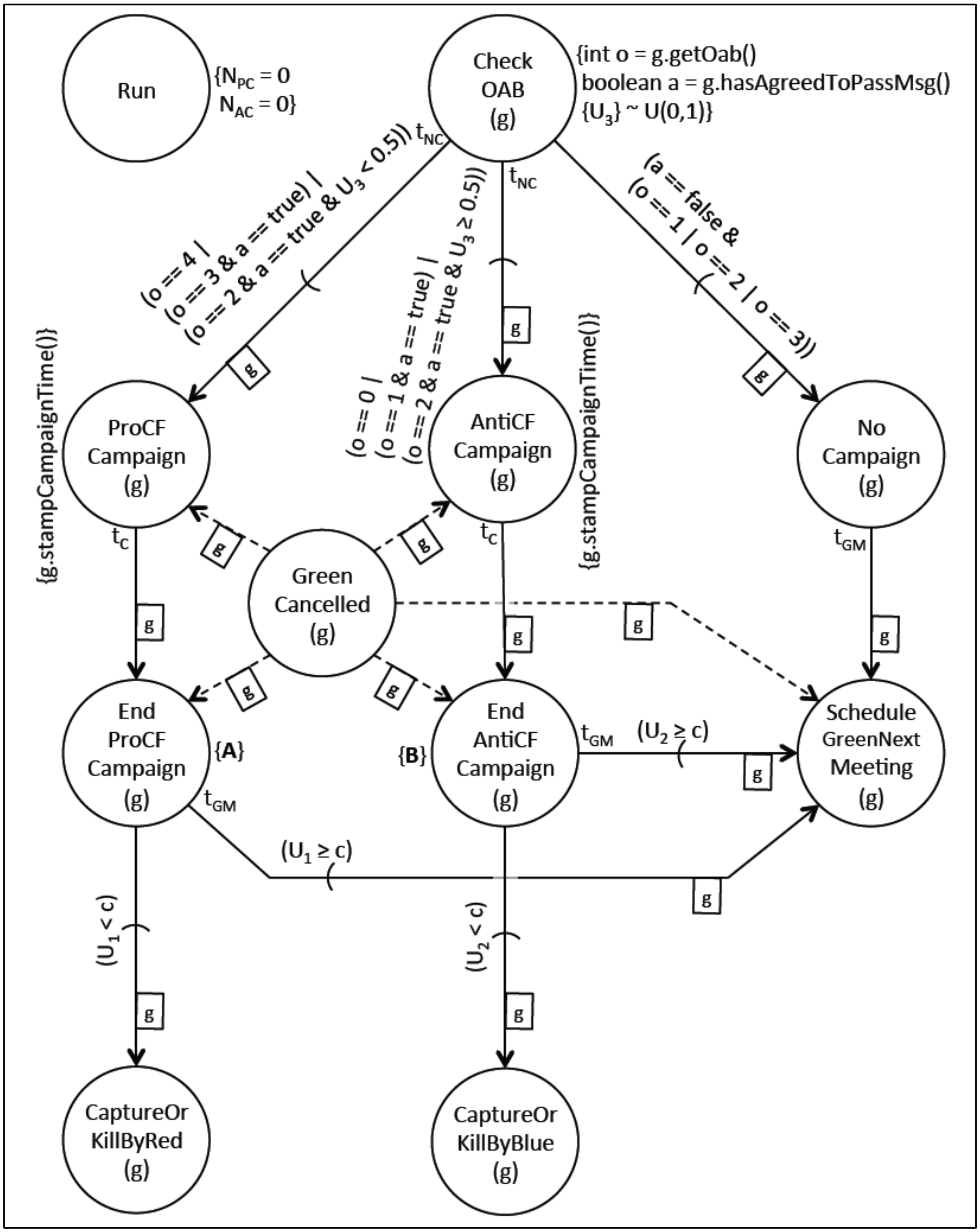


Figure 19. *Campaign* event graph (part 1)

A
{N_{PC}++
{U_1} ~ U(0,1)
double c = behaviorEquation3(p_{BCKR}, g.getCampaignElapsedTime())}

B
{N_{AC}++
{U_2} ~ U(0,1)
double c = behaviorEquation3(p_{BCKB}, g.getCampaignElapsedTime())}

Figure 20. *Campaign* event graph (part 2)

The *Run* event initializes the two state variables to zero.

The *CheckOAB* event checks a Green player's OAB to see if he will campaign or not. It takes a *GreenPlayer* agent as its local parameter. It draws a random uniform number between 0 and 1. If the local Green player has an OAB equal to 4, has an OAB equal to 3 and has agreed to pass a message, or has an OAB equal to 2, has agreed to pass a message, and the uniform draw is less than 0.5, it schedules a *ProCFCampaign* event with a time delay pulled from {t_{NC}}, passing along the local Green player. If the local Green player has an OAB equal to 0, has an OAB equal to 1 and has agreed to pass a message, or has an OAB equal to 2, has agreed to pass a message, and the uniform draw is greater than or equal to 0.5, it schedules an *AntiCFCampaign* event with a time delay pulled from {t_{NC}}, passing along the local Green player. If the Green player has not agreed to pass a message and his OAB equals 1, 2, or 3, it schedules a *NoCampaign* event, passing along the local Green player.

The *ProCFCampaign* event simulates a Green player starting his pro-coalition force campaign. It takes a *GreenPlayer* agent as its local parameter. It stamps the campaign start time for the local Green player. It then schedules an *EndProCFCampaign* event with a time delay pulled from {t_C}, passing along the local Green player.

The *AntiCFCampaign* event simulates a Green player starting his anti-coalition force campaign. It takes a *GreenPlayer* agent as its local parameter. It stamps the campaign start time for the local Green player. It then schedules an *EndAntiCFCampaign* event with a time delay pulled from $\{t_C\}$, passing along the local Green player.

The *NoCampaign* event simulates a Green player not campaigning. It takes a *GreenPlayer* agent as its local parameter. It schedules a *ScheduleGreenNextMeeting* event with a time delay pulled from $\{t_{GM}\}$, passing along the local Green player.

The *EndProCFCampaign* event simulates a Green player ending his pro-coalition force campaign. It takes a *GreenPlayer* agent as its local parameter. It increments N_{PC} by one. It then draws a random uniform number between 0 and 1. It calculates the probability that the local Green player is captured or killed by a Red player by using p_{BCKR} and the time spent in the campaign in behavior Equation 3 (Figure 3). If the random uniform draw is less than the calculated capture or kill by Red probability, it schedules a *CaptureOrKillByRed* event, passing along the local Green player. If the random uniform draw is greater than or equal to the calculated capture or kill by Red probability, it schedules a *ScheduleGreenNextMeeting* event with a time delay pulled from $\{t_{GM}\}$, passing along the local Green player.

The *EndAntiCFCampaign* event simulates a Green player ending his anti-coalition force campaign. It takes a *GreenPlayer* agent as its local parameter. It increments N_{AC} by one. It then draws a random uniform number between 0 and 1. It calculates the probability that the local Green player is captured or killed by a Blue player by using p_{BCKB} and the time spent in the campaign in behavior Equation 3 (Figure 3). If the random uniform draw is less than the calculated capture or kill by Blue probability, it schedules a *CaptureOrKillByBlue* event, passing along the local Green player. If the random uniform draw is greater than or equal to the calculated capture or kill by Blue probability, it schedules a

ScheduleGreenNextMeeting event with a time delay pulled from $\{t_{GM}\}$, passing along the local Green player.

The *ScheduleGreenNextMeeting* event simulates a Green player scheduling his next arrival for a KLE. It takes a *GreenPlayer* agent as its local parameter.

The *CaptureOrKillByRed* event simulates a Green player being captured or killed by a Red player. It takes a *GreenPlayer* agent as its local parameter.

The *CaptureOrKillByBlue* event simulates a Green player being captured or killed by a Blue player. It takes a *GreenPlayer* agent as its local parameter.

The *GreenCanceled* event simulates a Green player replacement that is no longer needed in the model. It takes a *GreenPlayer* agent as its local parameter. It cancels the *ProCFCampaign*, *AntiCFCampaign*, *EndProCFCampaign*, *EndAntiCFCampaign*, and *ScheduleGreenNextMeeting* events for the local Green player.

11. CaptureOrKill

The *CaptureOrKill* component handles whether a Green player will be captured or killed by a Blue player or Red player following a KLE or campaign. It has four input parameters. The parameter p_{CB} is the probability that a Blue player captures a Green player. One minus p_{CB} then is the probability that a Blue player kills him. The parameter p_{CR} is the probability that a Red player captures a Green player. One minus p_{CR} then is the probability that a Red player kills him. The parameter p_{DCB} is the probability that a Green player decreases his OAB given a capture by a Blue player. The parameter p_{ICR} is the probability that a Green player increases his OAB given a capture by a Red player.

The *CaptureOrKill* component has four state variables. The variables N_{BC}, N_{BK}, N_{RC}, and N_{RK} track the number of Green players captured by Blue players, killed by Blue players, captured by Red players, and killed by Red players, respectively.

Parameters and state variables for the *CaptureOrKill* component are summarized in Table 13.

Parameter	Description
p_{CB}	probability of capture by Blue player
$1 - p_{CB}$	probability of kill by Blue player
p_{CR}	probability of capture by Red player
$1 - p_{CR}$	probability of kill by Red player
p_{DCB}	probability of OAB decrease given capture by Blue player
p_{ICR}	probability of OAB increase given capture by Red player
State Variable	**Description**
N_{BC}	number of Green player captures by Blue players (initialized to 0)
N_{BK}	number of Green player kills by Blue players (initialized to 0)
N_{RC}	number of Green player captures by Red players (initialized to 0)
N_{RK}	number of Green player kills by Red players (initialized to 0)

Table 13. *CaptureOrKill* parameters and state variables

The event graph for the *CaptureOrKill* component is shown in Figures 21 and 22.

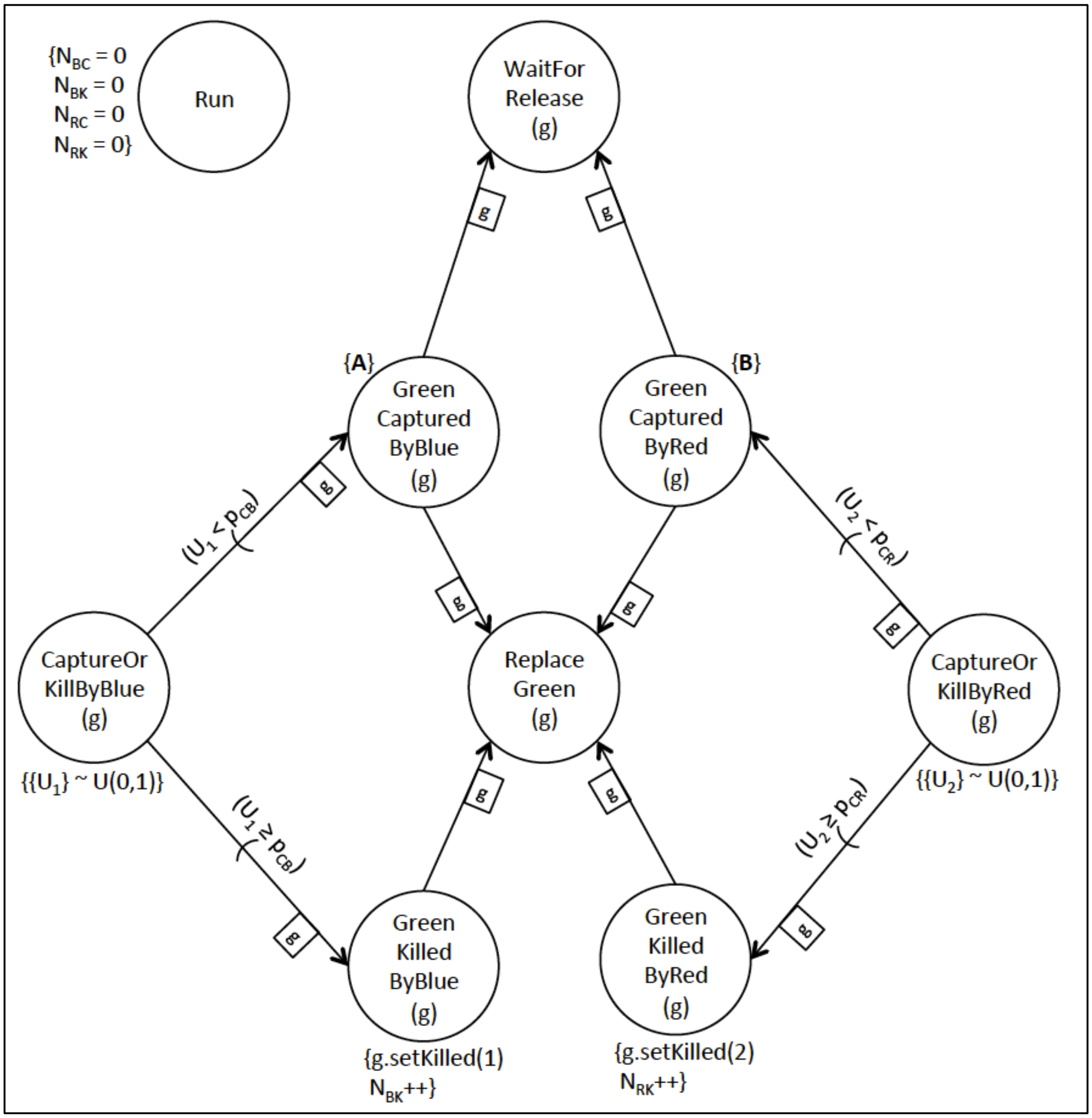


Figure 21. *CaptureOrKill* event graph (part 1)

A

{g.setCaptured(1)

N_{BC}++

g.setOab(behaviorEquation5(g.getOab(), p_{DCB}, 0))}

B

{g.setCaptured(2)

N_{RC}++

g.setOab(behaviorEquation5(g.getOab(), 0, p_{ICR}))}

Figure 22. *CaptureOrKill* event graph (part 2)

The *Run* event initializes all state variables to zero.

The *CaptureOrKillByBlue* event sees whether a Green player will be captured or killed by a Blue player. It takes a *GreenPlayer* agent as its local parameter. It draws a random uniform number between 0 and 1. If the random uniform draw is less than p_{CB}, it schedules a *GreenCapturedByBlue* event, passing along the local Green player. If the random uniform draw is greater than or equal to p_{CB}, it schedules a *GreenKilledByBlue* event, passing along the local Green player.

The *GreenCapturedByBlue* event simulates a Green player being captured by a Blue player. It takes a *GreenPlayer* agent as its local parameter. It sets the captured status of the local Green player as if he was captured by a Blue player and increments N_{BC} by one. It calculates the Green player's new OAB by using his current OAB, D = p_{DCB}, and I = 0 in behavior Equation 5 (Figure 5). It then schedules a *ReplaceGreen* event and a *WaitForRelease* event, passing along the local Green player to both.

The *GreenKilledByBlue* event simulates a Green player being killed by a Blue player. It takes a *GreenPlayer* agent as its local parameter. It sets the killed status of the local Green player as if he was killed by a Blue player, and it increments N_{BK} by one. It then schedules a *ReplaceGreen* event, passing along the local Green player.

The *CaptureOrKillByRed* event sees whether a Green player will be captured or killed by a Red player. It takes a *GreenPlayer* agent as its local parameter. It draws a random uniform number between 0 and 1. If the random uniform draw is less than p_{CR}, it schedules a *GreenCapturedByRed* event, passing along the local Green player. If the random uniform draw is greater than or equal to p_{CR}, it schedules a *GreenKilledByRed* event, passing along the local Green player.

The *GreenCapturedByRed* event simulates a Green player being captured by a Red player. It takes a *GreenPlayer* agent as its local parameter. It sets the captured status of the local Green player as if he was captured by a Red player and increments N_{RC} by one. It calculates the Green player's new OAB by using his current OAB, D = 0, and I = p_{ICR} in behavior Equation 5 (Figure 5). It then schedules a *ReplaceGreen* event and a *WaitForRelease* event, passing along the local Green player to both.

The *GreenKilledByRed* event simulates a Green player being killed by a Red player. It takes a *GreenPlayer* agent as its local parameter. It sets the killed status of the local Green player as if he was killed by a Red player, and it increments N_{RK} by one. It then schedules a *ReplaceGreen* event, passing along the local Green player.

The *ReplaceGreen* event simulates a Green player being replaced by another Green player after being captured or killed. It takes a *GreenPlayer* agent as its local parameter.

The *WaitForRelease* event simulates a Green player awaiting his release after being captured. It takes a *GreenPlayer* agent as its local parameter.

12. Release

The *Release* component represents the releasing of Green players after being captured. It has one input parameter. It requires a random distribution representing the stream of times that Green players are released ({t_{RL}}).

Parameters for the *Release* component are summarized in Table 14.

Parameter	Description
{t_{RL}}	stream of Green player release times

Table 14. *Release* parameters

The event graph for the *Release* component is shown in Figure 23.

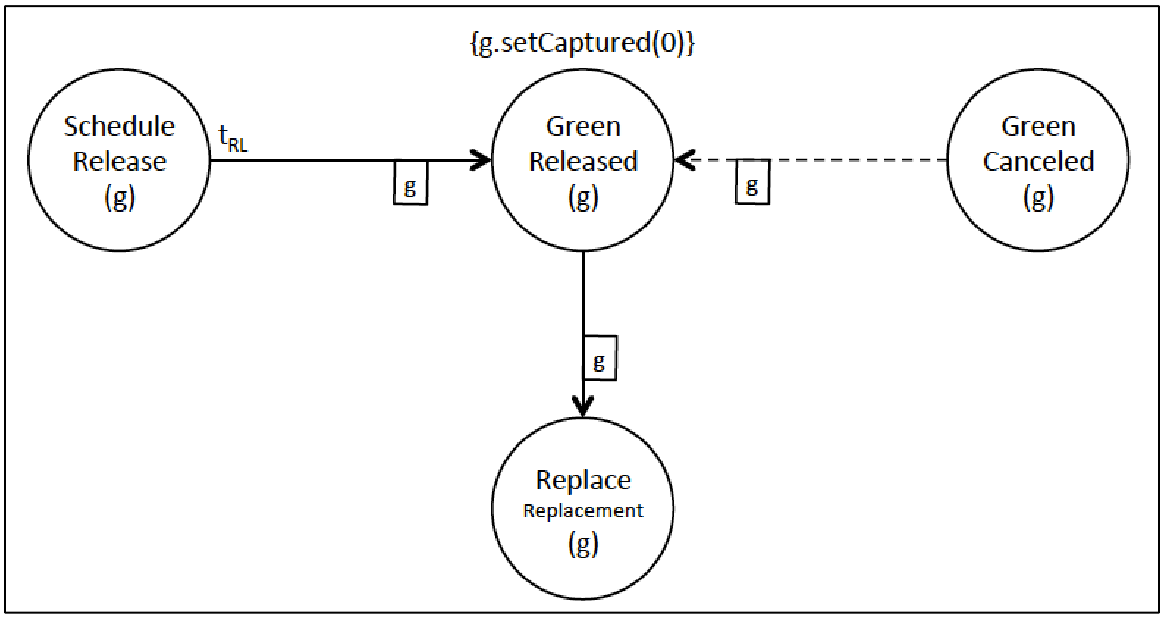


Figure 23. *Release* event graph

The *ScheduleRelease* event simulates a Green player waiting for his release after being captured. It takes a *GreenPlayer* agent as its local parameter. It schedules a *GreenReleased* event with a time delay pulled from {t_{RL}}, passing along the local Green player.

The *GreenReleased* event simulates a Green player being released. It takes a *GreenPlayer* agent as its local parameter. It resets the captured status of the local Green player to show that he is no longer captured. It then schedules a *ReplaceReplacement* event, passing along the local Green player.

The *ReplaceReplacement* event simulates a Green player taking control back from his replacement. It takes a *GreenPlayer* agent as its local parameter.

The *GreenCanceled* event simulates a Green player replacement that is no longer needed in the model. It takes a *GreenPlayer* agent as its local parameter. It cancels the *GreenReleased* event for the local Green player.

13. HandleReplacements

The *HandleReplacements* component handles the replacing of Green players when they are captured, killed, or released. It has 13 input parameters. It requires one random distribution representing the stream of times that Green players schedule their next arrival for another KLE ($\{t_{GM}\}$). The parameters p_C, p_{KLK}, p_{TK}, and p_{RK} are the same parameters from the *CreatePlayers* component representing the probabilities that a Green player is corrupt, has key leader critical knowledge, has threat critical knowledge, and has resource critical knowledge, respectively. The parameters p_{LKB} and p_{HKB} represent the probabilities of a Green replacement having a lower or higher OAB, respectively, than the Green player that is killed by a Blue player; the sum of these two must be less than or equal to 1. The parameters p_{LKR} and p_{HKR} represent the probabilities of a Green replacement having a lower or higher OAB, respectively, than the Green player that is killed by a Red player; the sum of these two must be less than or equal to 1. The parameters p_{LCB} and p_{HCB} represent the probabilities of a Green replacement having a lower or higher OAB, respectively, than the Green player that is captured by a Blue player; the sum of these two must be less than or equal to 1. The parameters p_{LCR} and p_{HCR} represent the probabilities of a Green replacement having a lower or higher OAB, respectively, than the Green player that is captured by a Red player; the sum of these two must be less than or equal to 1.

The *HandleReplacements* component has one state variable. The variable *c* represents a list to hold captured Green players that have a replacement in the model.

Parameters, parameter constraints, and state variables for the *HandleReplacements* component are summarized in Table 15.

Parameter	Description
$\{t_{GM}\}$	stream of Green player next meeting times
p_C	probability of Green player being corrupt
p_{KLK}	probability of Green player having key leader knowledge
p_{TK}	probability of Green player having threat knowledge
p_{RK}	probability of Green player having resource knowledge
p_{LKB}	probability of replacement having lower OAB when Green player killed by Blue player
p_{HKB}	probability of replacement having higher OAB when Green player killed by Blue player
p_{LKR}	probability of replacement having lower OAB when Green player killed by Red player
p_{HKR}	probability of replacement having higher OAB when Green player killed by Red player
p_{LCB}	probability of replacement having lower OAB when Green player captured by Blue player
p_{HCB}	probability of replacement having higher OAB when Green player captured by Blue player
p_{LCR}	probability of replacement having lower OAB when Green player captured by Red player
p_{HCR}	probability of replacement having higher OAB when Green player captured by Red player
Parameter Constraint	
$p_{LKB} + p_{HKB} \leq 1$	
$p_{LKR} + p_{HKR} \leq 1$	
$p_{LCB} + p_{HCB} \leq 1$	
$p_{LCR} + p_{HCR} \leq 1$	
State Variable	**Description**
c	list to hold captured Green players that have replacement

Table 15. *HandleReplacements* parameters, parameter constraints, and state variables

The event graph for the *HandleReplacements* component is shown in Figure 24.

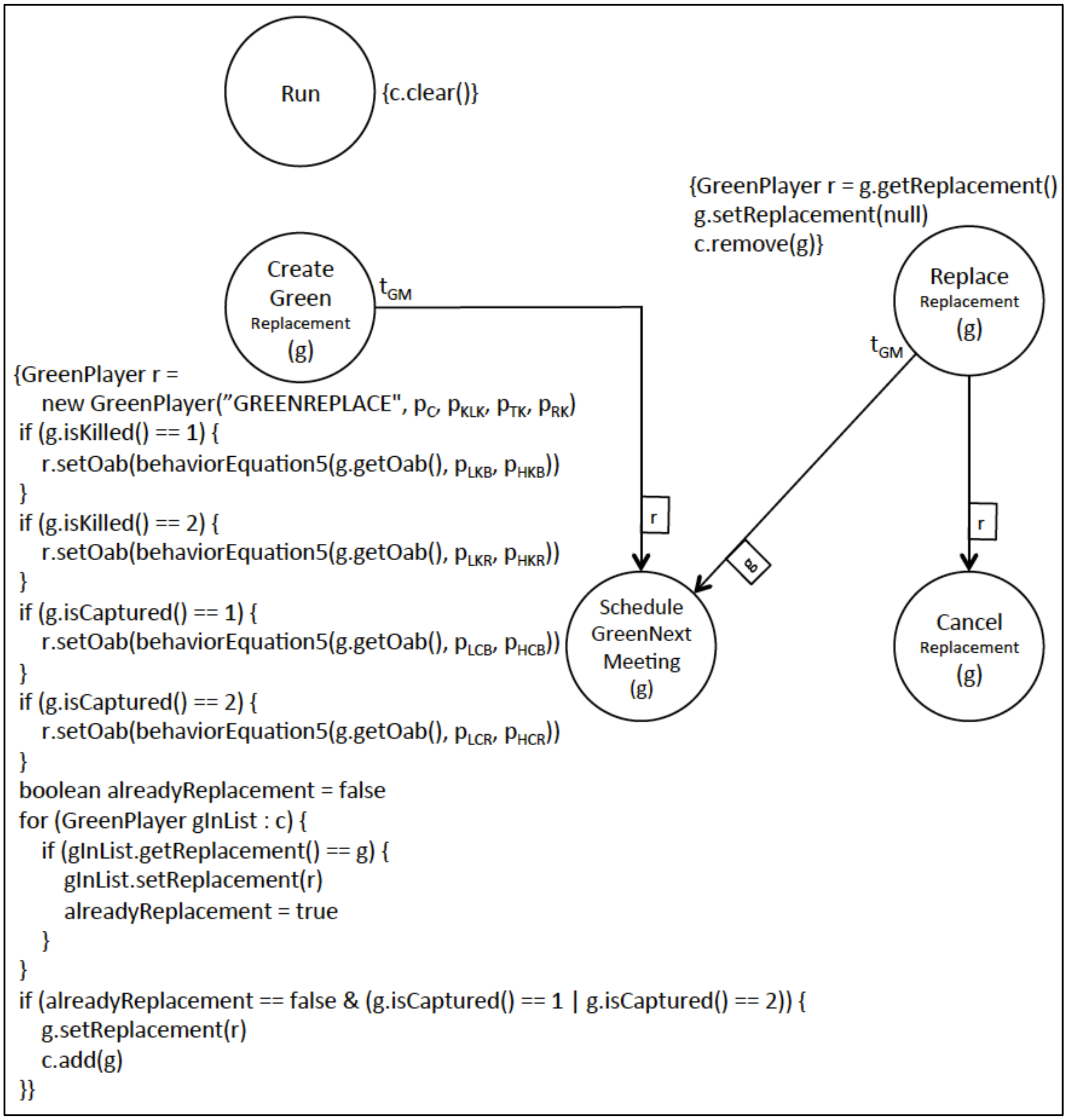


Figure 24. *HandleReplacements* event graph

The *Run* event clears *c*.

The *CreateGreenReplacement* event simulates a Green replacement being added to the model when a Green player is captured or killed. It takes a *GreenPlayer* agent as its local parameter. It creates a new *GreenPlayer* agent that is the replacement for the local Green player. If a Blue player kills the local Green player, it calculates the replacement's OAB by using the Green player's OAB, D = p_{LKB}, and I = p_{HKB} in behavior Equation 5 (Figure 5). If the local Green player is killed by a Red player, it calculates the replacement's OAB by using the Green player's OAB, D = p_{LKR}, and I = p_{HKR} in behavior Equation 5 (Figure 5). If the local Green player is captured by a Blue player, it calculates the replacement's OAB by using the Green player's OAB, D = p_{LCB}, and I = p_{HCB} in behavior Equation 5 (Figure 5). If the local Green player is captured by a Red player, it calculates the replacement's OAB by using the Green player's OAB, D = p_{LCR}, and I = p_{HCR} in behavior Equation 5 (Figure 5). Then it checks the state variable, *c*, to determine if the local Green player that is killed or captured is already a replacement for another captured Green player. If this is the case, it sets the newly created replacement as the replacement for the Green player in *c*. Then, if the local Green player is not already a replacement and is captured, it sets the newly created replacement as his replacement and is added to *c*. Lastly, it schedules a *ScheduleGreenNextMeeting* event with a time delay pulled from $\{t_{GM}\}$, passing along the created replacement.

The *ReplaceReplacement* event simulates a Green player being released and his replacement being no longer needed in the model. It takes a *GreenPlayer* agent as its local parameter. It takes the Green replacement assigned to the released Green player and assigns this replacement to a local *GreenPlayer* agent variable. It resets the replacement of the released Green player to null, and then it removes the released Green player from *c*. Lastly, it schedules a *ScheduleGreenNextMeeting* event with a time delay pulled from $\{t_{GM}\}$, passing along the released Green player, and it schedules a *CancelReplacement* event, passing along the replacement.

The *ScheduleGreenNextMeeting* event simulates a Green player scheduling his next arrival for a KLE. It takes a *GreenPlayer* agent as its local parameter.

The *CancelReplacement* event simulates a Green player replacement no longer being needed in the model. It takes a *GreenPlayer* agent as its local parameter.

E. COMPONENT LISTENING STRUCTURE AND ADAPTERS OF KLE MODEL

The various components of the KLE Model are connected together as shown in Figure 25. The various adapters in the KLE Model are listed in Table 16. When one of the listed events for a given component is executed, the respective listening component schedules the appropriate event. For more information on connecting event graphs using listeners and adapters, see Buss (2011).

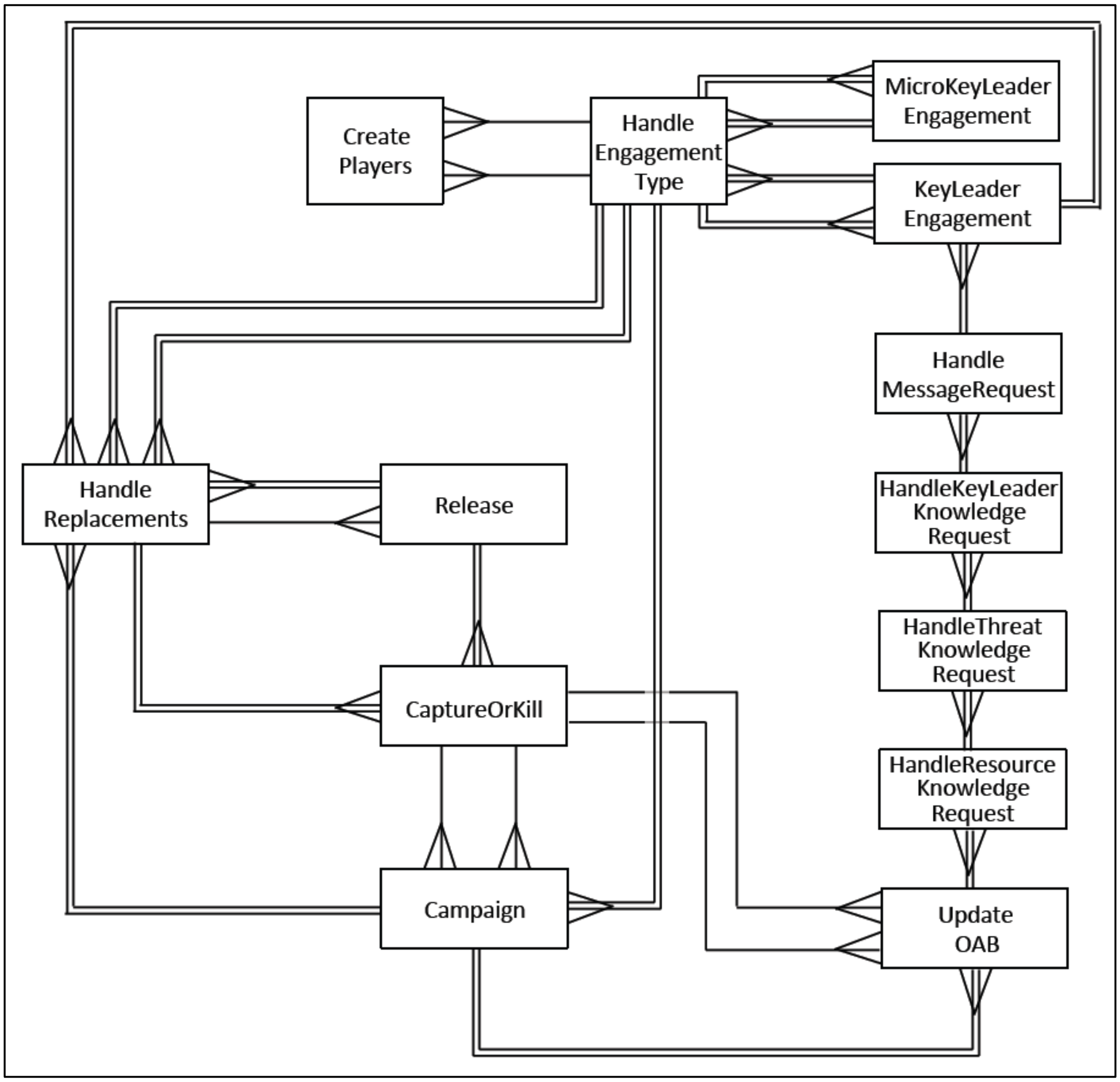


Figure 25. KLE Model component listening structure

EXECUTED COMPONENT/EVENT		LISTENING COMPONENT/EVENT	
Component	Event	Component	Event
Create Players	BluePlayer Arrival	Handle EngagementType	BluePlayer Arrival
Create Players	GreenPlayer Arrival	Handle EngagementType	GreenPlayer Arrival
MicroKeyLeader Engagement	ScheduleBlue NextMeeting	Handle EngagementType	BluePlayer Arrival
KeyLeader Engagement	ScheduleBlue NextMeeting	Handle EngagementType	BluePlayer Arrival
Campaign	ScheduleGreen NextMeeting	Handle EngagementType	GreenPlayer Arrival
Handle Replacements	ScheduleGreen NextMeeting	Handle EngagementType	GreenPlayer Arrival
Handle Replacements	Cancel Replacement	Handle EngagementType	Green Canceled
Handle EngagementType	BlueReady ForMicroKLE	MicroKeyLeader Engagement	Start MicroKLE
Handle EngagementType	SendPlayers ToKLE	KeyLeader Engagement	Start KLE
Handle Replacements	Cancel Replacement	KeyLeader Engagement	Green Canceled
KeyLeader Engagement	Handle Requests	Handle MessageRequest	StartMessage Request
Handle MessageRequest	EndMessage Request	HandleKeyLeader KnowledgeRequest	StartKeyLeader KnowledgeRequest
HandleKeyLeader KnowledgeRequest	EndKeyLeader KnowledgeRequest	HandleThreat KnowledgeRequest	StartThreat KnowledgeRequest
HandleThreat KnowledgeRequest	EndThreat KnowledgeRequest	HandleResource KnowledgeRequest	StartResource KnowledgeRequest
HandleResource KnowledgeRequest	EndResource KnowledgeRequest	UpdateOAB	CheckGreen Status
Update OAB	Campaign	Campaign	Check OAB
Handle Replacements	Cancel Replacement	Campaign	Green Canceled
Update OAB	CaptureOr KillByBlue	CaptureOrKill	CaptureOr KillByBlue
Update OAB	CaptureOr KillByRed	CaptureOrKill	CaptureOr KillByRed
Campaign	CaptureOr KillByBlue	CaptureOrKill	CaptureOr KillByBlue
Campaign	CaptureOr KillByRed	CaptureOrKill	CaptureOr KillByRed
Capture OrKill	WaitFor Release	Release	Schedule Release
Handle Replacements	Cancel Replacement	Release	Green Canceled
Capture OrKill	Replace Green	Handle Replacements	CreateGreen Replacement
Release	Replace Replacement	Handle Replacements	Replace Replacement

Table 16. KLE Model adapters

THIS PAGE INTENTIONALLY LEFT BLANK

III. DESIGN OF EXPERIMENTS

Chapter III begins with a brief discussion on the use of random number generators in the Key Leader Engagement (KLE) Model. Then we discuss what model input parameters were not varied and those that were, including their low and high values. We talk briefly about the nearly orthogonal and balanced mixed design of experiments that was utilized for further analysis of the model. The chapter ends with a small discussion on the scenario replication and three scenarios (one-week, nine-weeks, and one-year) that were executed.

A. RANDOM NUMBER GENERATION

Two random number streams are used to run the KLE Model. One generator creates random seeds for each design point run, and the other generator utilizes the seed to generate random numbers that are needed when running the model components. The two random number streams used when running the KLE Model both use the Mersenne Twister MT 19937 pseudorandom number generator (Wikipedia 2012).

B. HANDLING OF INPUT PARAMETERS

1. Static Parameters

The input parameters not varied in the design of experiments are those associated with numbers of players and all streams of time. The constant values assigned to these static parameters are best-guess estimates derived from military and civilian analysts at TRAC-Monterey that best coincide with what can be expected during a tactical wargame (TWG) using an Afghanistan scenario. The time streams are all triangle distributed (minimum, maximum, mode).

Table 17 lists the parameters that are not varied, the component(s) they are found in, and their associated values.

Parameter	Component(s)	Value
N_{BP}	CreatePlayers	4
N_{GP}	CreatePlayers	17
$\{t_{NM}\}$	HandleEngagementType	Triangle(0.25,1,0.5)
$\{t_{NK}\}$	HandleEngagementType	Triangle(1,168,48)
$\{t_{RG}\}$	HandleEngagementType	Triangle(1,168,48)
$\{t_{BM}\}$	HandleEngagementType MicroKeyLeaderEngagement KeyLeaderEngagement	Triangle(24,72,48)
$\{t_{M}\}$	MicroKeyLeaderEngagement	Triangle(0.1,0.5,0.2)
$\{t_{K}\}$	KeyLeaderEngagement	Triangle(1,6,4)
$\{t_{NC}\}$	Campaign	Triangle(1,168,24)
$\{t_{C}\}$	Campaign	Triangle(0.5,48,3)
$\{t_{BM}\}$	Campaign HandleReplacements	Triangle(24,336,96)
$\{t_{RL}\}$	Release	Triangle(24,8760,2160)

Table 17. Static model parameters and their values

2. Dynamic Parameters and NOB Mixed Design

The input parameters varied in the design of experiments are those associated with probabilities and probability factors. Probabilities not associated with OAB changes or assignments are varied from 0 to 1. Probabilities associated with OAB changes and assignments are varied from 0 to 0.5 due to the decrease/increase or lower/higher pairings used in behavior Equation 5 (Figure 5). Probability factors are varied from 0 to 0.2. The baseline probabilities, p_{BCKB} and p_{BCKR}, are varied from 0 to 0.0208 due to the parameter restriction that these two individually multiplied by the maximum campaign time (48) must be less than or equal to 1.

Tables 18 and 19 list the parameters that are varied, the component(s) they are found in, and their associated low and high values.

Parameter	Component(s)	Low Value	High Value
p_I	CreatePlayers KeyLeaderEngagement	0	1
p_C	CreatePlayers HandleReplacements	0	1
p_{KLK}	CreatePlayers HandleKeyLeaderKnowledgeRequest HandleReplacements	0	1
p_{TK}	CreatePlayers HandleThreatKnowledgeRequest HandleReplacements	0	1
p_{RK}	CreatePlayers HandleResourceKnowledgeRequest HandleReplacements	0	1
pf_{RG}	HandleEngagementType	0	0.2
pf_{NS}	HandleEngagementType	0	0.2
pf_{CK}	MicroKeyLeaderEngagement	0	0.2
pf_{BT}	HandleMessageRequest HandleKeyLeaderKnowledgeRequest HandleThreatKnowledgeRequest HandleResourceKnowledgeRequest	0	0.2
pf_{HR}	HandleMessageRequest HandleKeyLeaderKnowledgeRequest HandleThreatKnowledgeRequest HandleResourceKnowledgeRequest	0	0.2
pf_{HRI}	HandleMessageRequest HandleKeyLeaderKnowledgeRequest HandleThreatKnowledgeRequest HandleResourceKnowledgeRequest	0	0.2
pf_{HRT}	HandleMessageRequest HandleKeyLeaderKnowledgeRequest HandleThreatKnowledgeRequest HandleResourceKnowledgeRequest	0	0.2

Table 18. Dynamic model parameters and their values (part 1)

Parameter	Component(s)	Low Value	High Value
p_{D0}	UpdateOAB	0	0.5
p_{I0}	UpdateOAB	0	0.5
p_{DI}	UpdateOAB	0	0.5
p_{II}	UpdateOAB	0	0.5
p_{DT}	UpdateOAB	0	0.5
p_{IT}	UpdateOAB	0	0.5
p_{BCKB}	UpdateOAB Campaign	0	0.0208
p_{BCKR}	UpdateOAB Campaign	0	0.0208
p_{CB}	CaptureOrKill	0	1
p_{CR}	CaptureOrKill	0	1
p_{DCB}	CaptureOrKill	0	0.5
p_{ICR}	CaptureOrKill	0	0.5
p_{LKB}	HandleReplacements	0	0.5
p_{HKB}	HandleReplacements	0	0.5
p_{LKR}	HandleReplacements	0	0.5
p_{HKR}	HandleReplacements	0	0.5
p_{LCB}	HandleReplacements	0	0.5
p_{HCB}	HandleReplacements	0	0.5
p_{LCR}	HandleReplacements	0	0.5
p_{HCR}	HandleReplacements	0	0.5

Table 19. Dynamic model parameters and their values (part 2)

The design is constructed using the 512-design point nearly orthogonal and balanced (NOB) mixed design spreadsheet of Vieira (2012). The result is a nearly orthogonal Latin hypercube (NOLH) since all parameters in this design are continuous-valued. For more details about the properties or application of NOLH designs, see Kleijnen et al. (2005) or Sanchez et al. (2012). For more details about NOB designs, which can also handle discrete-valued factors with limited numbers of levels, see Vieira et al. (2011, 2012).

C. SCENARIO REPLICATION

In order to assist with the code verification efforts of the KLE Model, three different scenarios are used. Within the model, one unit of simulated time represents one hour of real time. The first scenario looks at short-term effects within the model and warm-up period issues; the model is run for 168 time units (hours) to represent the span of a week. The second looks at mid-range effects; the model is run for 1,512 time units to represent the span of nine weeks, the typical run time for a TWG. The third looks at long-term effects and convergence issues; the model is run for 8,760 time units to represent the span of a year. Additionally, each design point for each scenario is replicated 200 times to collect summary statistics for analysis, and to allow for the possibility of examining the variances as well as the means of the output responses of interest.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. KLE MODEL ANALYSIS

Chapter IV begins with a short description of the model output data. The analysis begins with a look at the significant input parameters used to build regression metamodels and partition tree models to help verify the KLE Model execution. We then look at the output summary statistics to gain insights into the ranges and the variability of the output responses. Finally, some discussion on the number of micro-KLEs response is presented given the apparently anomalous behavior of this output variable.

A. OUTPUT DATA

The outputs analyzed in the KLE Model are associated with all the countable state variables within the model components; these are of interest as they correlate to the outputs analyzed during a TRAC tactical wargame (TWG). For each design point, we collect the final values of the state variables for all 200 replications. We then output the mean, standard deviation, minimum value, and maximum value for the 200 replications.

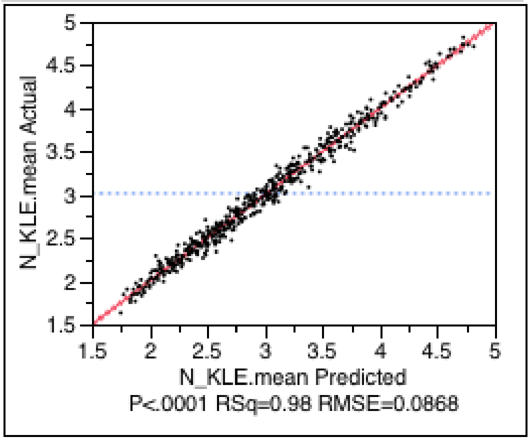
B. SIGNIFICANT INPUT PARAMETERS AND MODEL VERIFICATION

In order to explore the significant input factors for each of the output responses, and subsequently help verify the expected functionality of the KLE Model, we first derive second-order regression metamodels that best fit each output response. A stepwise regression control with a minimum Bayesian information criterion stopping rule is used to find the input parameters that are significant in predicting the responses. These parameters (after removing less significant terms) are then used to fit the regression metamodel using standard least squares. From the sorted parameter estimates, we can see which input parameters are the most significant. Second, we create partition tree models with up to 20 splits if needed to identify the most significant input parameters for each response; this helps verify the significant parameters derived in the regression metamodels. The statistical software JMP® Pro 9.0 was used to create these regression metamodels and partition tree models.

To get an idea of the regression metamodels and partition tree models created, we use the number of KLEs output response as an example. Starting with the metamodels, Figures 26, 27, and 28 show the second-order regression metamodels for the number of KLEs in the one-week, nine-week, and one-year scenarios, respectively. All three metamodels show an F-statistic p-value of less than 0.0001, indicating statistical significance in all cases; all three have relatively high R-squared values (greater than 0.9); and all three metamodels have terms that are statistically significant (t-statistic p-values less than 0.01). We remark that with such a large data set, statistical significance is necessary but not sufficient for including terms in the metamodels. In some cases, we have eliminated terms with p-values less than 0.01 in the interests of parsimony, when their inclusion leads to very little improvement in a metamodel's R-squared value. Figure 27 illustrates this phenomenon; if we simplified the metamodel even further by eliminating the four interaction terms with p-values between 0.0003 and 0.0020, the R-squared value would drop only slightly (from 0.9898 to 0.9886). The simplified metamodel is preferable. Similar simplifications could be made for the one-year metamodel.

From these regression metamodels, we see that the renege probability factor (pf_{RG}) and the no-show probability factor (pf_{NS}) are the two most significant parameters for the one-week and nine-week scenarios and within the top three for the one-year scenario. These two parameters are the primary factors of whether a Blue player engages a Green player, and as model runtime increases, these factors remain significant, which is what we were looking for in the KLE Model execution. The figures also exhibit the increasing complexity of the metamodels as runtime increases due to the greater influence of cross-component effects, which is expected.

Actual by Predicted Plot



Summary of Fit

RSquare	0.984757
RSquare Adj	0.984667
Root Mean Square Error	0.086814
Mean of Response	3.017539
Observations (or Sum Wgts)	512

Analysis of Variance

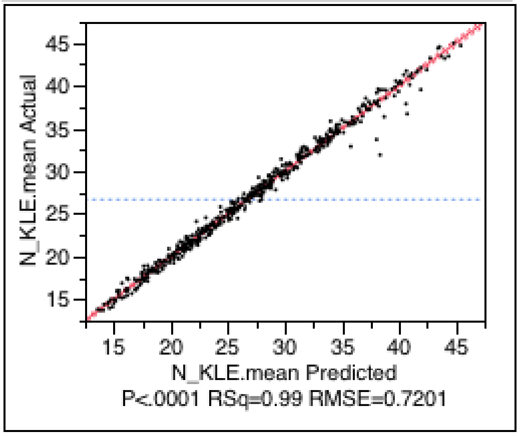
Source	DF	Sum of Squares	Mean Square	F Ratio
Model	3	247.33762	82.4459	10939.29
Error	508	3.82863	0.0075	**Prob > F**
C. Total	511	251.16625		<.0001*

Sorted Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
pf_RG	-8.533547	0.066527	-128.3	<.0001*
pf_NS	-8.435853	0.0664	-127.0	<.0001*
(pf_RG-0.1)*(pf_NS-0.1)	40.781917	1.162246	35.09	<.0001*

Figure 26. Number of KLEs regression metamodel (1 week)

Actual by Predicted Plot



Summary of Fit

RSquare	0.989796
RSquare Adj	0.989593
Root Mean Square Error	0.72012
Mean of Response	26.67324
Observations (or Sum Wgts)	512

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	10	25202.059	2520.21	4859.890
Error	501	259.805	0.52	**Prob > F**
C. Total	511	25461.864		<.0001*

Sorted Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
pf_NS	-84.88342	0.550977	-154.1	<.0001*
pf_RG	-85.14389	0.552799	-154.0	<.0001*
(pf_RG-0.1)*(pf_NS-0.1)	401.85299	9.667813	41.57	<.0001*
p_KLK	-2.383182	0.110567	-21.55	<.0001*
pf_HR	-9.853123	0.552565	-17.83	<.0001*
(p_KLK-0.5)*(pf_HR-0.1)	-26.35962	1.904628	-13.84	<.0001*
(p_KLK-0.5)*(pf_RG-0.1)	6.8493926	1.867203	3.67	0.0003*
(p_KLK-0.5)*(pf_NS-0.1)	6.9780912	1.936586	3.60	0.0003*
(pf_NS-0.1)*(pf_HR-0.1)	30.764916	9.286546	3.31	0.0010*
(pf_RG-0.1)*(pf_HR-0.1)	29.482919	9.469376	3.11	0.0020*

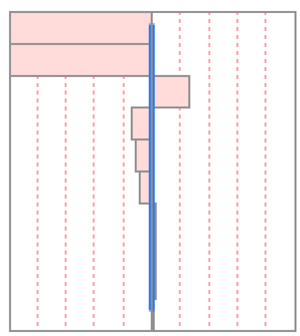
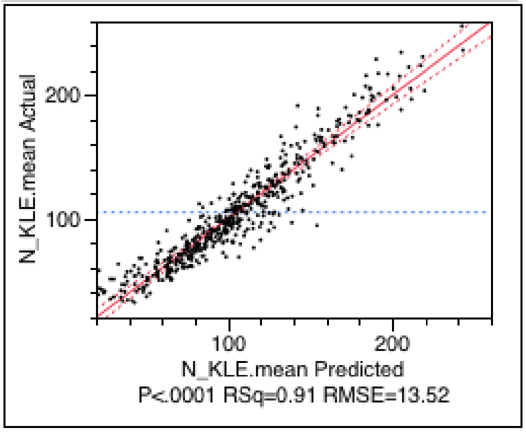


Figure 27. Number of KLEs regression metamodel (9 weeks)

Actual by Predicted Plot



Summary of Fit

RSquare	0.907715
RSquare Adj	0.904346
Root Mean Square Error	13.52019
Mean of Response	105.2959
Observations (or Sum Wgts)	512

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	18	886404.52	49244.7	269.3975
Error	493	90118.26	182.8	**Prob > F**
C. Total	511	976522.78		<.0001*

Sorted Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
p_KLK	-86.86023	2.091336	-41.53	<.0001*
pf_NS	-293.6074	10.43467	-28.14	<.0001*
pf_HR	-292.9832	10.41999	-28.12	<.0001*
pf_RG	-288.615	10.38332	-27.80	<.0001*
(p_KLK-0.5)*(pf_NS-0.1)	446.60665	36.70415	12.17	<.0001*
(p_KLK-0.5)*(pf_RG-0.1)	369.44452	35.37955	10.44	<.0001*
(p_KLK-0.5)*(pf_HR-0.1)	-353.6441	36.07581	-9.80	<.0001*
(pf_NS-0.1)*(pf_HR-0.1)	1474.9089	176.3731	8.36	<.0001*
(pf_RG-0.1)*(pf_HR-0.1)	1338.5144	178.5741	7.50	<.0001*
p_C	-12.1191	2.083207	-5.82	<.0001*
p_I	-9.853334	2.091881	-4.71	<.0001*
pf_BT	-46.77026	10.38876	-4.50	<.0001*
pf_HRT	-44.15599	10.45169	-4.22	<.0001*
(p_C-0.5)*(pf_HR-0.1)	154.45205	37.13474	4.16	<.0001*
(pf_HR-0.1)*(pf_HRI-0.1)	710.98815	185.4616	3.83	0.0001*
pf_HRI	-36.66194	10.4566	-3.51	0.0005*
(p_KLK-0.5)*(pf_HRT-0.1)	-112.5208	36.64736	-3.07	0.0023*
(pf_RG-0.1)*(pf_NS-0.1)	523.22212	182.6806	2.86	0.0044*

Figure 28. Number of KLEs regression metamodel (1 year)

Figures 29, 30, and 31 show the partition tree models for the number of KLEs in the one-week, nine-week, and one-year scenarios, respectively. These trees back-up what was discovered in the regression metamodels, especially the initial split using the probability of having key leader critical knowledge (p_{KLK}) in the one-year scenario (Figure 31), which corresponds to the parameter's significance in the one-year regression metamodel (Figure 28). For simplicity, only three or four levels within the partition trees are displayed. The resulting R-squared values are lower than they were for the corresponding regression metamodels. Even so, looking at the output in both ways is useful, since responses with discontinuities in the results may fit much better with partition tree models than with regression metamodels. Partition trees are also sometimes easier graphs for communicating with decision makers (Sanchez et al. 2012).

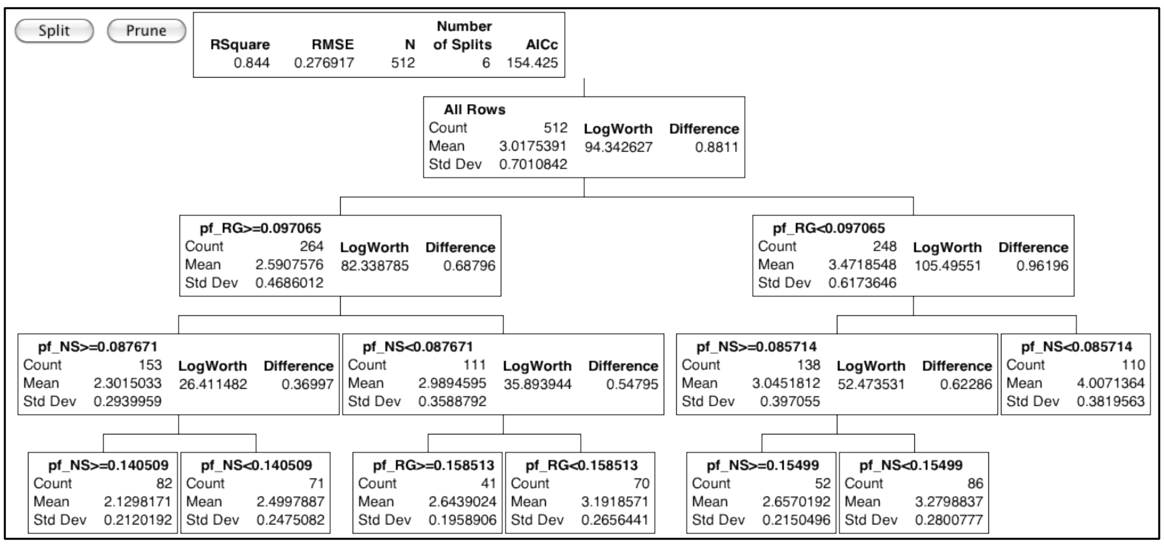


Figure 29. Number of KLEs partition tree model (1 week)

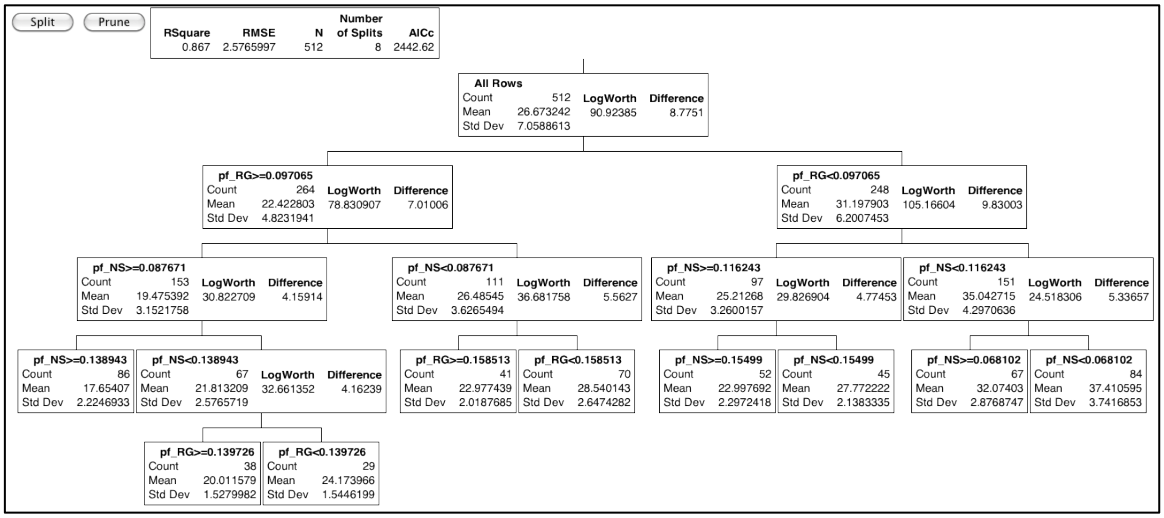


Figure 30. Number of KLEs partition tree model (9 weeks)

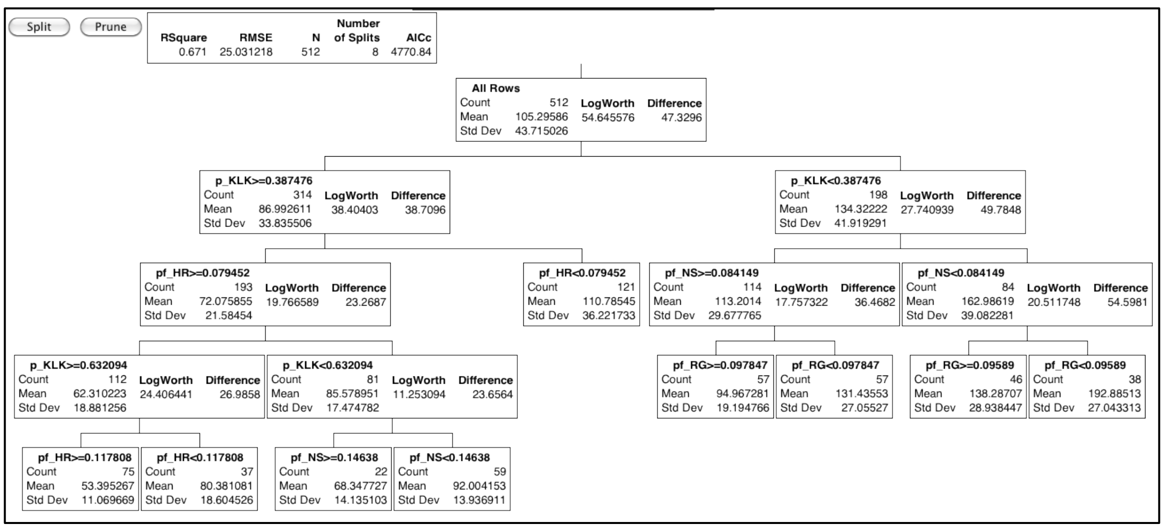


Figure 31. Number of KLEs partition tree model (1 year)

Having discussed the techniques used to derive the second-order regression metamodels and partition tree models, Tables 20, 21, and 22 show which input parameters are the top three most significant when building the metamodels (denoted by #) and tree models (denoted by &) for the output responses for the one-week scenario, nine-week scenario, and one-year scenario, respectively. Two count columns show the total number of times an input parameter is one of the top three most significant in the regression metamodels and likewise for the partition tree models.

Input Param	Regr Meta-Models Count	Part Trees Count	N_{MKLE}	N_{TKG}	N_{KLE}	N_{HRM}	N_{HRK}	N_{HRT}	N_{HRR}	N_{PC}	N_{AC}	N_{BC}	N_{BK}	N_{RC}	N_{RK}
p_{I}	0	0													
p_{C}	0	0													
p_{KLK}	6	5					# &	# &	# &	# &				#	# &
p_{TK}	1	1						# &							
p_{RK}	1	1							# &						
pf_{RG}	6	4			# &	# &				#	# &	# &	#		
pf_{NS}	4	6			# &	# &	# &			&	# &		&		
pf_{CK}	1	1		# &											
pf_{BT}	0	0													
pf_{HR}	6	6				# &	# &	# &	# &	# &	# &				
pf_{HRI}	0	0													
pf_{HRT}	0	0													
p_{DO}	0	0													
p_{IO}	0	0													
p_{DI}	0	0													
p_{II}	0	0													
p_{DT}	0	0													
p_{IT}	0	0													
p_{BCKB}	2	2										# &	# &		
p_{BCKR}	2	2												# &	# &
p_{CB}	2	2										# &	# &		
p_{CR}	2	2												# &	# &
p_{DCB}	0	0													
p_{ICR}	0	1												&	
p_{LKB}	0	0													
p_{HKB}	0	0													
p_{LKR}	0	0													
p_{HKR}	0	0													
p_{LCB}	0	0													
p_{HCB}	0	0													
p_{LCR}	0	0													
p_{HCR}	0	0													

= parameter included in regression metamodel that is one of top three most significant

& = parameter included in partition tree model that is one of top three most significant

Table 20. Top three significant input parameters (1 week).

Input Param	Regr Meta-Models Count	Part Trees Count	N_{MKLE}	N_{TKG}	N_{KLE}	N_{HRM}	N_{HRK}	N_{HRT}	N_{HRR}	N_{PC}	N_{AC}	N_{BC}	N_{BK}	N_{RC}	N_{RK}
p_I	0	0													
p_C	0	0													
p_{KLK}	9	9	# &	# &	#	&	# &	# &	# &	# &				# &	# &
p_{TK}	1	1						# &							
p_{RK}	1	1							# &						
pf_{RG}	7	6	&	# &	# &	#	# &			#	# &	&	#		
pf_{NS}	5	6	# &		# &	# &				&	# &	#	&		
pf_{CK}	0	1		&											
pf_{BT}	0	0													
pf_{HR}	8	6	#	#		# &	# &	# &	# &	# &	# &				
pf_{HRI}	0	0													
pf_{HRT}	0	0													
p_{DO}	0	0													
p_{IO}	0	0													
p_{DI}	0	0													
p_{II}	0	0													
p_{DT}	0	0													
p_{IT}	0	0													
p_{BCKB}	2	2										# &	# &		
p_{BCKR}	2	2												# &	# &
p_{CB}	2	2										# &	# &		
p_{CR}	2	2												# &	# &
p_{DCB}	0	0													
p_{ICR}	0	0													
p_{LKB}	0	0													
p_{HKB}	0	0													
p_{LKR}	0	0													
p_{HKR}	0	0													
p_{LCB}	0	0													
p_{HCB}	0	0													
p_{LCR}	0	0													
p_{HCR}	0	0													

= parameter included in regression metamodel that is one of top three most significant

& = parameter included in partition tree model that is one of top three most significant

Table 21. Top three significant input parameters (9 weeks)

Input Param	Regr Meta-Models Count	Part Trees Count	N_{MKLE}	N_{TKG}	N_{KLE}	N_{HRM}	N_{HRK}	N_{HRT}	N_{HRR}	N_{PC}	N_{AC}	N_{BC}	N_{BK}	N_{RC}	N_{RK}
p_I	0	0													
p_C	0	0													
p_{KLK}	10	11	# &	# &	# &	# &	# &	# &	# &	# &	&			# &	# &
p_{TK}	1	1						# &							
p_{RK}	1	1							# &						
pf_{RG}	7	3	# &		#	# &	#				# &	#	#		
pf_{NS}	3	4	#		# &		&				# &	&			
pf_{CK}	1	1		# &											
pf_{BT}	0	0													
pf_{HR}	6	7	&	# &	&	# &	# &	# &	#	# &					
pf_{HRI}	0	0													
pf_{HRT}	0	0													
p_{DO}	1	2								# &			&		
p_{IO}	0	1							&						
p_{DI}	0	0													
p_{II}	0	0													
p_{DT}	0	0													
p_{IT}	1	0									#				
p_{BCKB}	2	2										# &	# &		
p_{BCKR}	2	2												# &	# &
p_{CB}	2	2										# &	# &		
p_{CR}	2	2												# &	# &
p_{DCB}	0	0													
p_{ICR}	0	0													
p_{LKB}	0	0													
p_{HKB}	0	0													
p_{LKR}	0	0													
p_{HKR}	0	0													
p_{LCB}	0	0													
p_{HCB}	0	0													
p_{LCR}	0	0													
p_{HCR}	0	0													

= parameter included in regression metamodel that is one of top three most significant

& = parameter included in partition tree model that is one of top three most significant

Table 22. Top three significant input parameters (1 year)

In all three scenarios, we see that p_{KLK}, pf_{RG}, pf_{NS}, and pf_{HR} are the input parameters that are considered most significant the most times (counts highlighted in green) for both regression metamodels and partition tree models. The renege probability factor (pf_{RG}) and the no-show probability factor (pf_{NS}) make intuitive sense as these dictate whether a Green player ultimately shows up and partakes in a KLE, and KLEs are the driving force for most of the model outputs. The honoring requests probability factor (pf_{HR}) also makes intuitive sense as many actions that occur after KLEs depend on whether the Green player honored the various Blue player requests.

The probability of having key leader critical knowledge (p_{KLK}) seems peculiar as to why it is so important in predicting the various outputs; for instance, what does the number of pro-coalition force campaigns have to do with whether or not a Green player has critical knowledge on other key leaders? If a Green player has key leader critical knowledge, he can be incentivized or threatened to give this knowledge to a Blue player during a KLE. If he is incentivized or threatened, this can more significantly affect whether or not his OAB is updated following a KLE. This in turn impacts whether or not he will conduct a pro-coalition force campaign. Likewise, if the Green player does not have key leader critical knowledge, he is never incentivized or threatened, and so his OAB is less likely to change and the impact on conducting a pro-coalition force campaign is reduced. This effect-tracing through the various components applies to all the output responses.

Using Tables 21, 22, and 23, we can verify the functionality of the KLE Model and confirm that it worked properly over a large range of inputs. The number of micro-KLEs output is anomalous and is discussed in more detail in section D. The number of times knowledge is gained during micro-KLEs is expected to be linked to the chance of knowledge probability factor (pf_{CK}), and the metamodels and tree models support this fact. The number of KLEs is discussed in the example at the beginning of this section, and those analytical models support the expected behavior of the KLE Model.

For the honoring request outputs (pass message, provide key leader critical knowledge, provide threat critical knowledge, and provide resource critical knowledge), we expect pf_{RG} and pf_{NS} to be important (we cannot honor requests during KLEs if we do not attend KLEs), as well as pf_{HR}. Additionally, for the three critical knowledge-related outputs, we expect the probabilities of having said knowledge (p_{KLK}, p_{TK}, and p_{RK}) to help predict the respective responses, and they show up in the analytical models with high significance.

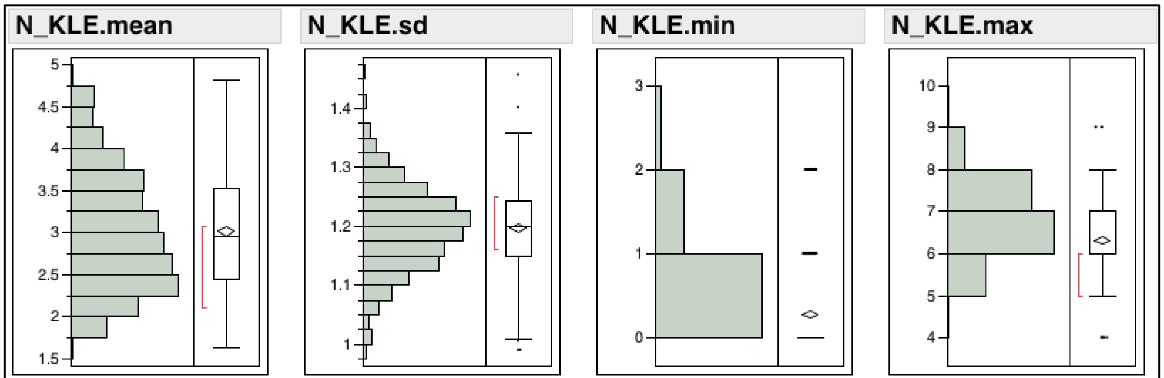
For the pro- and anti-coalition force campaign outputs, we expect pf_{RG} and pf_{NS} to be important (we cannot campaign following KLEs if we do not attend KLEs), as well as pf_{HR} since honoring or not honoring requests leads to potential incentives and threats that can impact the OAB updating following a KLE; the OAB directly impacts what type of campaign will occur. Once again, these factors show up in the analytical models with high significance.

The last set of outputs (the capture and kill outputs) verify the capturing and killing functionality by using the baseline probabilities of capture or kill by Blue players (p_{BCKB}) or by Red players (p_{BCKR}) and the probabilities of capturing vice killing by Blue players (p_{CB}) or by Red players (p_{CR}). We expect the respective Blue player probabilities to be significant when predicting captures and kills by Blue players, and likewise for the Red player probabilities. The metamodels and tree models support this fact in all cases.

C. SUMMARY STATISTICS ANALYSIS

Using JMP®, the distributions of the means, standard deviations, minimums, and maximums are attained for each output response per scenario. The goal is to gain insights into what the KLE Model can provide regarding issues such as variability or outliers. These snapshots include histograms, outlier boxplots, quantile data, and moment data. The summary statistics can be used (along with the histograms) to qualitatively assess whether the output is reasonable or not.

Once again we use the number of KLEs response as our example, and the distributions can be seen in Figures 32, 33, and 34 for the one-week, nine-week, and one-year scenarios, respectively. We see that the various data are well-distributed but with some skewness, especially in the mean and minimum histograms for all three runtimes. Note that there is no reason to expect that the distribution of the design point means should be symmetric, since the design point results arise from different combinations of inputs. In this example, there are no significant outliers. Using the number of KLEs mean statistics, and just using plus or minus one standard deviation from its mean, we expect our model to produce 2 to 4 KLEs in one week, 20 to 34 KLEs over nine weeks, and 62 to 149 KLEs over one year. This appears to scale nicely as runtime increases and so this range of values seems reasonable. Even when we look at the minimum and maximum numbers of KLEs experienced in all three scenarios, getting these minimums and maximums as results is reasonable also.



Quantiles

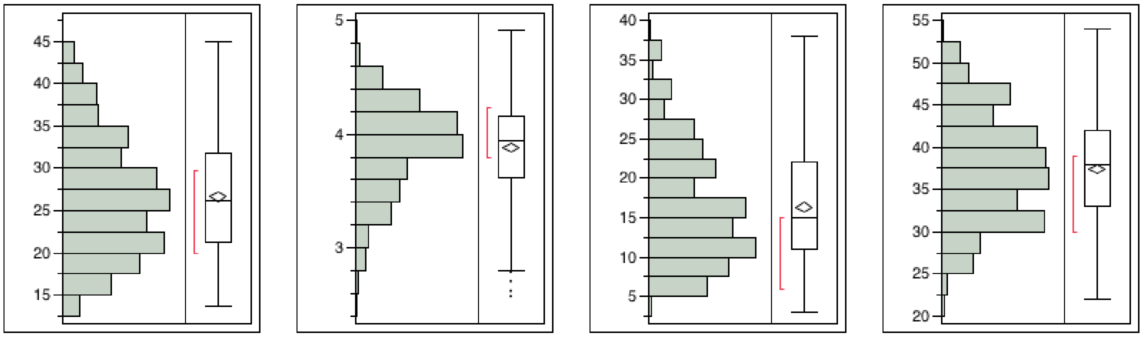
		N_KLE.mean	N_KLE.sd	N_KLE.min	N_KLE.max
100.0%	maximum	4.82	1.45726	2	9
99.5%		4.73718	1.40151	2	8.435
97.5%		4.52438	1.33325	2	8
90.0%		4.002	1.28905	1	7
75.0%	quartile	3.52875	1.24318	0	7
50.0%	median	2.955	1.19859	0	6
25.0%	quartile	2.455	1.14951	0	6
10.0%		2.17	1.11164	0	5
2.5%		1.87913	1.04985	0	5
0.5%		1.76065	0.99824	0	4
0.0%	minimum	1.63	0.98928	0	4

Moments

	N_KLE.mean	N_KLE.sd	N_KLE.min	N_KLE.max
Mean	3.0175391	1.1971577	0.28125	6.3183594
Std Dev	0.7010842	0.0708696	0.5336064	0.8613322
Std Err Mean	0.0309838	0.003132	0.0235823	0.0380659
Upper 95% Mean	3.0784104	1.2033109	0.3275802	6.3931442
Lower 95% Mean	2.9566677	1.1910045	0.2349198	6.2435745
N	512	512	512	512

Figure 32. Number of KLEs summary statistics (1 week)

	N_KLE.mean	N_KLE.sd	N_KLE.min	N_KLE.max



Quantiles

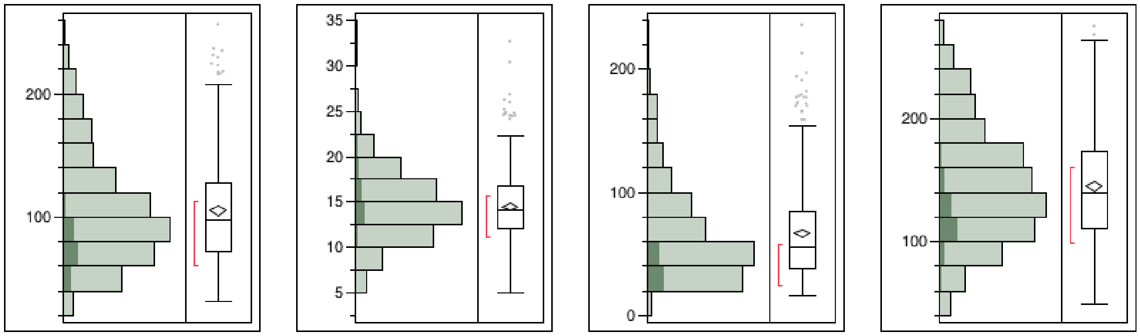
Quantile		N_KLE.mean	N_KLE.sd	N_KLE.min	N_KLE.max
100.0%	maximum	44.975	4.91438	38	54
99.5%		44.6002	4.72368	37	51
97.5%		41.5797	4.513	33	50
90.0%		36.626	4.33614	26	46
75.0%	quartile	31.7088	4.16233	22	42
50.0%	median	26.17	3.94194	15	38
25.0%	quartile	21.28	3.61776	11	33
10.0%		18.002	3.32107	8	29
2.5%		15.013	2.96161	6	25.825
0.5%		13.7824	2.66713	4.565	22.565
0.0%	minimum	13.705	2.57135	3	22

Moments

Moment	N_KLE.mean	N_KLE.sd	N_KLE.min	N_KLE.max
Mean	26.673242	3.8818726	16.34375	37.433594
Std Dev	7.0588613	0.3944533	7.3749855	6.3952866
Std Err Mean	0.3119605	0.0174325	0.3259314	0.2826344
Upper 95% Mean	27.286125	3.9161209	16.98408	37.988862
Lower 95% Mean	26.060359	3.8476244	15.70342	36.878325
N	512	512	512	512

Figure 33. Number of KLEs summary statistics (9 weeks)

N_KLE.mean | N_KLE.sd | N_KLE.min | N_KLE.max



N_KLE.mean — Quantiles

100.0%	maximum	256.32
99.5%		235.666
97.5%		206.398
90.0%		168.359
75.0%	quartile	128.385
50.0%	median	97.77
25.0%	quartile	72.7087
10.0%		55.3835
2.5%		40.8741
0.5%		33.5412
0.0%	minimum	32.03

Moments

Mean	105.29586
Std Dev	43.715026
Std Err Mean	1.9319494
Upper 95% Mean	109.0914
Lower 95% Mean	101.50032
N	512

N_KLE.sd — Quantiles

100.0%	maximum	32.6262
99.5%		28.3336
97.5%		24.4195
90.0%		19.2627
75.0%	quartile	16.8185
50.0%	median	14.0922
25.0%	quartile	12.049
10.0%		9.89553
2.5%		6.71971
0.5%		5.69634
0.0%	minimum	5.02327

Moments

Mean	14.507469
Std Dev	3.9302358
Std Err Mean	0.1736935
Upper 95% Mean	14.848711
Lower 95% Mean	14.166228
N	512

N_KLE.min — Quantiles

100.0%	maximum	236
99.5%		203.96
97.5%		173.7
90.0%		121
75.0%	quartile	85
50.0%	median	56
25.0%	quartile	38.25
10.0%		28
2.5%		21.825
0.5%		18.565
0.0%	minimum	16

Moments

Mean	66.726563
Std Dev	38.754274
Std Err Mean	1.7127131
Upper 95% Mean	70.091388
Lower 95% Mean	63.361737
N	512

N_KLE.max — Quantiles

100.0%	maximum	275
99.5%		265.74
97.5%		245.175
90.0%		215.7
75.0%	quartile	173.75
50.0%	median	140
25.0%	quartile	111
10.0%		87.3
2.5%		62.825
0.5%		52.13
0.0%	minimum	49

Moments

Mean	145.2207
Std Dev	46.811277
Std Err Mean	2.0687857
Upper 95% Mean	149.28508
Lower 95% Mean	141.15633
N	512

Figure 34. Number of KLEs summary statistics (1 year)

Table 23 lists the summary statistics for all of the output means across all three scenarios.

Output Response	1 week (mean/sd min/max)	9 weeks (mean/sd min/max)	1 year (mean/sd min/max)
N_{MKLE}	1/0 1/1	36.48/144.04 1/1879.44	7905.92/8840.24 1/34357.6
N_{TKG}	0.25/0.15 0/0.58	9.15/45.81 0/760.69	1987.56/2828.08 0/15503.4
N_{KLE}	3.02/0.7 1.63/4.82	26.67/7.06 13.71/44.98	105.3/43.72 32.03/256.32
N_{HRM}	1.14/0.54 0.06/2.76	9.28/4.29 0.67/23.67	30.13/17.79 3.01/118.73
N_{HRK}	0.57/0.46 0/2.42	4.46/3.27 0/15.37	11.74/5.39 0/20.78
N_{HRT}	0.42/0.34 0/1.74	3.51/2.86 0/15.9	12.34/11.97 0/75.15
N_{HRR}	0.42/0.34 0/2.38	3.3/2.67 0/19.91	11.04/10.62 0/70.56
N_{PC}	0.46/0.13 0.13/0.83	7.15/2.41 1.66/14.65	26.16/19.81 2.23/153.39
N_{AC}	0.23/0.13 0.01/0.75	4.66/2.56 0.54/15.24	26.13/15.36 3.56/107.63
N_{BC}	0.02/0.03 0/0.15	0.44/0.5 0/3.58	2.47/2.92 0/29.44
N_{BK}	0.84/0.51 0/2	0.45/0.51 0/3.37	2.43/2.76 0/18.5
N_{RC}	0.09/0.08 0/0.42	1.04/1.01 0/5.49	4.23/5.43 0/38.77
N_{RK}	0.08/0.08 0/0.4	1.05/1.02 0/5.27	4.03/4.83 0/35.89

Table 23. Summary statistics for output response means

For the number of micro-KLEs (N_{MKLE}) response, we observe that exactly one micro-KLE takes place during the one-week scenario. This is most likely due to the proportion of Blue players (4) to Green players (17) used in the scenarios; a Green player is almost always available to engage, so we never see more than (or less than) one micro-KLE. We also observe an exponential increase in the number of micro-KLEs as model runtime increases, which is discussed in section D. This exponential issue ties into the number of times knowledge is gained during micro-KLEs (N_{TKG}), but the problem is with the number of micro-KLEs only, as pf_{CK} is the driving force for N_{TKG}.

All of our responses are nonnegative. Some of their distributions are highly skewed, with standard deviations that are quite large relative to the means, which is why we report the minimum and maximum value along with the means and standard deviations. This still results in plausible ranges for all of the output variables (except N_{MKLE} and N_{TKG}) for all three scenarios, so our model is producing reasonable responses.

Only two of our design points produced significant outliers. One of these included the same outlier; they were the N_{MKLE} minimum boxplot and N_{TKG} minimum boxplot, both at nine weeks. This was associated with design point 443. All other design points produced only one micro-KLE and zero times knowledge gained, but the outlier values were 59 micro-KLEs and 24 times knowledge gained. The third outlier was found in the number of Blue captures minimum boxplot at one year, and it was associated with design point 63. Approximately 90% of the design points produced zero Blue captures, but this outlier value was 14. After looking at the input parameter values associated with these design points, no significant explanation was found for these three outliers and we attribute this to randomness within the model.

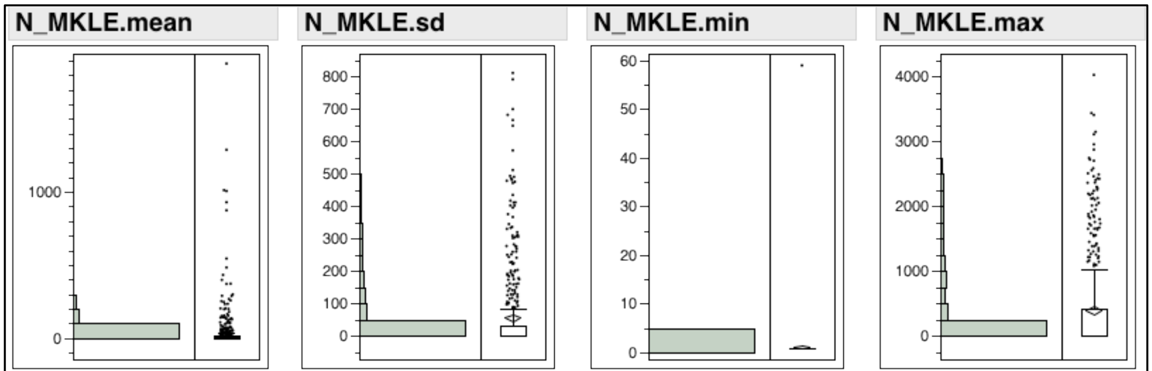
D. DISCUSSION ON NUMBER OF MICRO-KLES

After deriving regression metamodels and partition tree models for the number of micro-KLEs output response, we are able to decipher which input parameters are most significant in predicting number of micro-KLEs, but the analytical models themselves provide poor fits and poor explanations of the variability. In fact, we found that the number of micro-KLEs grows exponentially as scenario runtime increases. In the one-week scenario, we always had one micro-KLE occurring, but as we increase model runtime to nine weeks, we see a big jump in the mean number of micro-KLEs (Figure 35), and we experience exponential growth as we run the model for one-year (Figure 36).

This is one instance that might not show up as a problem if a single scenario time (nine weeks for instance) was used. A systematic exploration shows this anomaly compared to the other KLE Model output responses. After verifying that the KLE Model logic was sound and the implementation within Java was correct, the anomaly was found to be linked to the static input parameters governing the time a Blue player spends waiting for a micro-KLE and the time a Blue player spends in a micro-KLE.

From Table 17, these triangle-distributed time streams have very small modes compared to all the other time streams utilized in the model. The mode for the next scheduled micro-KLE time stream is 0.5, and the mode for the time spent in a micro-KLE time stream is 0.2. If there are no Green players available to engage, then a Blue player could do about 34 micro-KLEs a day. With four Blue players in the model, and assuming a one-year scenario, we could see upwards of 49,640 micro-KLEs in one-year combined.

The time streams were best-guess estimates from TRAC-Monterey analysts, so one solution is to think more carefully about what static time stream distribution is used for micro-KLEs. Another solution is that the micro-KLE functionality used in our model may require modifications (such as constraints on the total number of micro-KLEs that one agent can conduct over the course of a week, or the opportunity to “do nothing” rather than initiate a micro-KLE if no key leader is available) to meet TRAC’s needs before any incorporation into the CG Model.



Quantiles

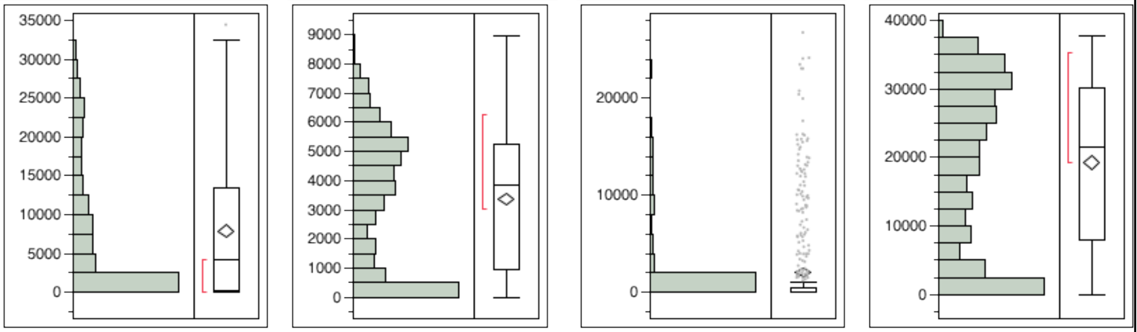
Quantile		N_MKLE.mean	N_MKLE.sd	N_MKLE.min	N_MKLE.max
100.0%	maximum	1879.44	811.746	59	4029
99.5%		1132.1	740.197	1	3423.62
97.5%		310.013	479.66	1	2519.4
90.0%		74.3665	203.814	1	1656.7
75.0%	quartile	6.19125	34.0482	1	413.75
50.0%	median	1	0	1	1
25.0%	quartile	1	0	1	1
10.0%		1	0	1	1
2.5%		1	0	1	1
0.5%		1	0	1	1
0.0%	minimum	1	0	1	1

Moments

	N_MKLE.mean	N_MKLE.sd	N_MKLE.min	N_MKLE.max
Mean	36.479268	57.433273	1.1132813	399.98047
Std Dev	144.03641	127.78772	2.5632621	744.35171
Std Err Mean	6.3655703	5.6474728	0.1132813	32.896009
Upper 95% Mean	48.985177	68.528396	1.3358355	464.60853
Lower 95% Mean	23.973358	46.338151	0.890727	335.3524
N	512	512	512	512

Figure 35. Number of micro-KLEs summary statistics (9 weeks)

N_MKLE.mean	N_MKLE.sd	N_MKLE.min	N_MKLE.max



Quantiles

		N_MKLE.mean	N_MKLE.sd	N_MKLE.min	N_MKLE.max
100.0%	maximum	34357.6	8988.39	26686	37779
99.5%		32398	8101.64	24034.2	37668.2
97.5%		28211.7	7350.48	16120.8	36560
90.0%		22980.9	6245.31	9507.8	34130.6
75.0%	quartile	13470.4	5259.32	440.5	30106.8
50.0%	median	4265.19	3831.48	1	21597.5
25.0%	quartile	303.501	931.738	1	7927
10.0%		13.4335	74.7203	1	783.4
2.5%		1	0	1	1
0.5%		1	0	1	1
0.0%	minimum	1	0	1	1

Moments

	N_MKLE.mean	N_MKLE.sd	N_MKLE.min	N_MKLE.max
Mean	7905.9202	3380.9514	2021.4473	19245.549
Std Dev	8840.2415	2312.1678	4717.1349	12063.153
Std Err Mean	390.68717	102.18434	208.46988	533.12108
Upper 95% Mean	8673.4709	3581.7045	2431.0108	20292.928
Lower 95% Mean	7138.3694	3180.1982	1611.8837	18198.17
N	512	512	512	512

Figure 36. Number of micro-KLEs summary statistics (1 year)

THIS PAGE INTENTIONALLY LEFT BLANK

V. WRAP-UP

We begin Chapter V by stating our conclusions as to what we accomplished by creating a KLE Model and analyzing that model. We then discuss the significant contributions that were made by conducting the research. We end with some discussion as to potential future research opportunities that stem from our research.

A. CONCLUSIONS

The primary goal of this research was to develop a discrete event simulation model for potential plug-in to the CG Model. This model would take the place of Nexus when analyzing KLEs by simplifying the Nexus code. We were able to show that a simple and understandable model can be built using Simkit that reasonably models those aspects of Nexus needed for the CG Model. Through the use of event graphs, we were able to represent the complexities of KLEs in a visually understandable way. In addition, by using discrete event simulation and event graphs, the KLE Model can be easily modified while still maintaining the desired functionality of the original model.

The purpose of the analysis was to test the KLE Model in order to verify that it works properly, and to gain an understanding of KLEs for areas of future research that can be pursued using this model. Various insights can be gathered from this research and analysis. Through the use of experimental design, we were able to adequately analyze what input parameters are most significant in the KLE Model and how these parameters verify the code implementation. Using the number of KLEs response as an example, we were also able to see through regression metamodels that output complexity increases with runtime as cross-component effects become influential. Our analysis identified four input parameters that show up most often in regression metamodels and partition tree models for the output variables, and showed that are also the most significant in the KLE Model. Three of these parameters made intuitive sense; the fourth, the probability of having key leader critical knowledge, can be shown to make sense as it has cross-component implications within the model. Lastly, we found that our model encountered difficulties modeling micro-KLEs, but the source of the problem was identified and properly addressed.

B. SIGNIFICANT CONTRIBUTIONS

The primary objective of this work is to enhance the CG Model in the highest priority areas of dynamic social network relationships and persuasion and influence (Jackson 2009). We sought to help satisfy the critical area requirements identified by the U.S. Army and U.S. Marine Corps. By incorporating those components of Nexus into the CG Model, this work has the potential to save the Army and Marine Corps time and money if and when the model becomes a wide-scale decision-making tool. This effort reduces long-term requirements for scenario file development and model maintenance. Lastly, this research provides a better understanding of key leader engagements and the part they play in cultural geography.

C. FUTURE RESEARCH OPPORTUNITIES

The KLE Model event graphs allow future researchers to identify where modifications and/or additions are necessary in order to achieve a desired outcome. Improvement in the functionality of the KLE Model can occur by expanding on the behavior modeling of Blue players and Green players. The behavior equations utilized are simple and easy to understand, but if found unsatisfactory, more complex, social theory-based equations can be applied in the model. Also, Red player actions were implied through various events, and future research could look at the feasibility of adding a Red player agent as a separate entity and analyzing outputs specific to its utilization.

This research ran the KLE Model as a closed-loop, stand-alone simulation. Future research may look into tailoring the KLE Model to the specifics of the CG Model. Then, by using the plug-and-play aspect of the CG Model, one could link the KLE Model up and see how the KLE Model outputs affect the general population, and how population behaviors as inputs affect the workings of the KLE Model.

The scenarios used in the analysis involved three distinct runtimes: one week, nine weeks, and one year. This enabled us to look at distinct differences in short-term, mid-range, and long-term model execution, but nothing in between. Future research might look at including model runtime as a parameter to further explore runtime effects on the output responses. Additionally, the numbers of Blue players and Green players were static parameters, as well as the streams of times used in the KLE Model. Future research could look at varying these aspects in a systematic way to study the effects of varying numbers of players and time streams.

Lastly, due to the large amount of data collected from running the model in the three scenarios over the 13 different output variables, this research made use of simple techniques to analyze the KLE Model. With more time, more advanced analytical techniques could be utilized to take a closer look at the data and extract any insights or relationships that were not shown in this research.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Baez, F. (2011). *Cultural geography model*. PowerPoint brief, unpublished, TRAC-Monterey.

Buss, A. (2011). *Discrete event simulation modeling*. Monterey, CA: Arnold Buss.

Caldwell, J. C., & Brown, R. (2011). *Social Impact Module (SIM) transition requirements*. Technical report, unpublished, TRAC-Monterey.

Conover, W. J. (1999). *Practical nonparametric statistics* (3rd ed.). New York, NY: John Wiley & Sons, Inc.

Duong, D. (n.d.). *Nexus and Indra*. PowerPoint brief, unpublished, TRAC-Monterey.

Jackson, L. A. (2009). *Cultural geography model enhancements for irregular warfare analysis*. Technical report, unpublished, TRAC-Monterey.

Kleijnen, J. P. C., Sanchez, S. M., Lucas, T. W., & Cioppa, T. M. (2005). A user's guide to the brave new world of designing simulation experiments. *INFORMS Journal on Computing*, *17*(3), 263–289.

McKenna, S., & Hampsey, R. (2010). The "COIN Warrior" waging influence: hints for the counterinsurgency (COIN) strategy in Afghanistan. *Special to Small Wars Journal*.

Sanchez, S. M., Lucas, T. W., Sanchez, P. J., Nannini, C. J., and Wan, H. (2012). Designing large scale simulation experiments, with applications to defense and homeland security. In *Design and analysis of experiments, special designs and applications (vol. 3)*, ed. Hinkelmann, K. Hoboken, New Jersey: Wiley, 413–442.

TRAC. (n.d.). *IW tactical wargame player's handbook: a manual for the irregular warfare tactical wargame*. Technical manual, unpublished, TRAC-White Sands.

Vieira, Jr., H. (2012). *NOB_Mixed_512DP_template_v1.xls design spreadsheet*. Retrieved April 1, 2012, from http://harvest.nps.edu

Vieira Jr., H., Sanchez, S. M., Kienitz, K. H., & Belderrain, M. C. N. (2011). Generating and improving orthogonal designs by using mixed integer programming. *European Journal of Operational Research*, *215*, 629-638.

Vieira, Jr., H., Sanchez, S. M., Kienitz, K. H., & Belderrain, M. C. N. (2012). Conducting trade-off analyses via simulation: efficient nearly orthogonal nearly balanced mixed designs. Working paper, Operations Research Department, Naval Postgraduate School, Monterey, CA.

Wikipedia. (2012). *Mersenne twister*. Retrieved April 13, 2012, from http://en.wikipedia.org/wiki/Mersenne_twister

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Arnold H. Buss
   Naval Postgraduate School
   Monterey, California

4. Susan M. Sanchez
   Naval Postgraduate School
   Monterey, California

5. TRAC-Monterey
   Naval Postgraduate School
   Monterey, California

6. Marine Corps Representative
   Naval Postgraduate School
   Monterey, California

7. Director, Training and Education, MCCDC, Code C46
   Quantico, Virginia

8. Director, Marine Corps Research Center, MCCDC, Code C40RC
   Quantico, Virginia

9. Marine Corps Tactical Systems Support Activity (Attn: Operations Officer)
   Camp Pendleton, California

10. Director, Studies and Analysis Division, MCCDC, Code C45
    Quantico, Virginia